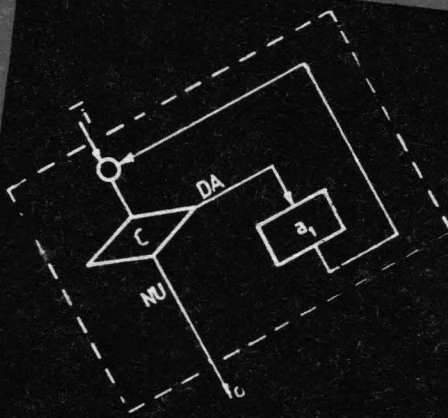


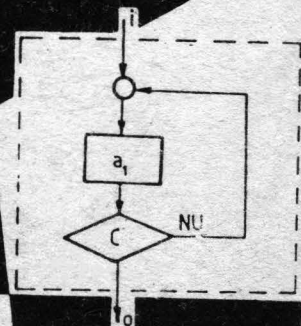
Să învățăm să programăm

Camelia Zaharia
Marian Zaharia

BASIC



EDITURA TEHNICĂ



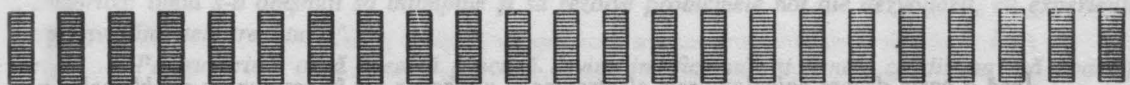
Marian Zaharia

Camelia Zaharia

INTRODUCERE

Problema "problema" nu este una de logica? Programarea
puncteaza in ordine a intrebărilor, nu-i revin alii

Constantin Noica - Jurnal de idei



SĂ ÎNVĂȚĂM SĂ PROGRAMĂM



In aceasta idee, lucrarea de față își propune să evidențieze importanța calității răspunsurilor
la întrebările CE și CUM, pentru un nivel concret, dar nu restrâns, acela al abordării aspectelor
legate de utilizarea sistemelor automate de calcul cu ajutorul unui limbaj de programare (BASIC), de
abordarea, formalizarea și programarea cât mai eficientă a soluționării corecte și rapide a diverselor
probleme.

Lucrări în domeniu, care să includă în cuprinsul lor - unul sau parțial - asemenea aspecte,
s-au scris foarte multe, dar adresate îndeosebi specialiștilor. Ceea ce aduce nou această carte este
stăruirea conștientă și într-un mod pedagogic, pe seară-gădușă de complexitate și informație,
pe o viață integratoare.

Plecând de la enunțul unei probleme - în general - lucrarea trece prin toate etapele logice ale
rezolvării ei - respectiv: mareaștruirea, descompoziția pe pași, conceperea algoritmului, descrierea
scrisă - ca apoi să facă introducerea în spațiul informatic al rezolvării și, respectiv, alcătuirea
programului. Prin parcurgerea acestor etape se urmărește formarea și dezvoltarea unei modalități
de abordare și gândire logice în soluționarea problemelor de matematică, dar și în



Editura Tehnică

București 1992

La capăt trebuie să obții o transfigurare. Transfigurarea mea e logica.

Problema "problemei" nu este una de logică? Programarea, punerea în ordine a întrebărilor, nu-i revin ei?

Constantin Noica - Jurnal de idei

Încă o carte despre calculatoarele electronice și utilizarea lor în rezolvarea problemelor pe care și le-a pus întotdeauna omul!

Nimic mai natural în epoca noastră în care, așa cum spunea un mare savant: "Ne înecăm în informații, dar sîntem infomețați de cunoaștere".

Și această foame de cunoaștere nu poate fi potolită decît prin astfel de cărți, cum este cea de față, pe care prestigioasa Editură Tehnică o pune generos la dispoziția unei largi categorii de cititori - elevi, studenți, profesori, specialiști în informatică.

Tot mai mulți oameni au nevoie de calculatorul electronic ca de o unealtă de muncă. Alții îl au ca partener de joacă sau de învățatură.

În această ultimă ipostază ni-l prezintă și nouă autorul, cunoscut informatician, cu o bogată experiență în domeniul programării calculatoarelor. Scopul declarat al cărții este să ne învețe cum să formulăm problemele pe care le avem de rezolvat, astfel încît ele să fie înțelese și rezolvate, mult mai repede și mai bine, de către calculator.

Este un demers îndrăzneț, dar experiența și talentul pedagogic ale autorului au produs o carte extrem de utilă, în primul rînd viitorilor informaticieni, tinerilor decîși să-și aleagă o profesie în lumea fascinantă a calculatoarelor electronice.

Conf. dr. Emil Scarlat



Editura Tehnică
București 1977

INTRODUCERE

"Una din dilemele condiției umane este că experiența și știința noastră se referă la trecut, în timp ce deciziile le luăm cu privire la viitor. Această dilemă existențială a generat o practică istorică: omul s-a obișnuit să întâmpine și să rezolve problemele noi ale dezvoltării, cu experiența și învățămintele trecutului".

Pentru prima oară această practică devine insuficientă; și pentru ca dilema să poată fi rezolvată, trebuie ca omul să poată învăța de la viitor în aceeași manieră în care a învățat și învățat de la trecut; iar pentru aceasta trebuie, în primul rând să știe atât să gândească logic, sistemic, cât și să beneficieze de un ajutor substanțial: calculatorul. Iată de ce, și această carte, pledează pentru a învăța să folosim - prin propriul nostru filtru - una din ultimele cuceriri ale omului în materie de "tehnică inteligentă", calculatorul electronic.

Existența cotidiană ne pune permanent în față probleme din ce în ce mai numeroase și mai complexe.

A știi CE SĂ FACI - este un prim pas în realizarea oricărui scop propus, a știi CUM SĂ FACI - este nu numai al doilea pas în ordinea firească, dar, mai ales, cheia reușitei, a îndeplinirii acestuia, echivalentul practic al încărcăturii magice a lui "SESAM DESCHIDE-TE".

Omul, de când există, a fost frământat de aceste întrebări. Găsind răspunsuri - mai bune sau mai puțin bune - a reușit să-și apropie elementele mediului înconjurător, să-și creeze actualul mediu de viață, a evoluat, a creat, dezvoltat și perfecționat în jurul său de la propria gândire, pînă la fabricarea unei gândiri similare lui, dar artificială, pentru mașinile sale, care să preia din ce în ce mai mult, nu numai sarcinile de rutină ci, mai nou, chiar elemente din activitatea de creație - care pînă de curînd i-a aparținut în întregime.

În această idee, lucrarea de față își propune să evidențieze importanța calității răspunsurilor la întrebările CE și CUM, pentru un nivel concret, dar nu restrîns, acela al abordării aspectelor legate de utilizarea sistemelor automate de calcul cu ajutorul unui limbaj de programare (BASIC), de abordarea, formalizarea și programarea cât mai eficientă a soluționării corecte și rapide a diverselor probleme.

Lucrări în domeniu, care să includă în cuprinsul lor - total sau parțial - asemenea aspecte, s-au scris foarte multe, dar adresate îndeosebi specialiștilor. Ceea ce aduce nou această carte este abordarea conținutului său într-un mod pedagogic, pe scară graduată de complexitate și informație, într-o viziune integratoare.

Plecînd de la enunțul unei probleme - în general - lucrarea trece prin toate etapele logice ale rezolvării ei - respectiv: teoretizarea, descompunerea pe pași, conceperea algoritmului, descrierea acestuia - ca apoi să facă introducerea în spațiul informatizat al rezolvării ei, respectiv alcătuirea programului. Prin parcurgerea acestor etape se urmărește formarea și dezvoltarea unei modalități de abordare și gândire logică nu numai în rezolvarea problemelor de matematică, dar și în

soluționarea oricăror tipuri de probleme cu care ne "ciocnim" în viața cotidiană.

De ce s-a ales BASIC, ca limbaj de programare pentru aplicațiile ce vin să concretizeze aspectele abordate? Pentru că este un limbaj relativ ușor de însușit atât pentru cei ce nu cunosc programare, cât și pentru cei familiarizați deja cu alte limbaje, cât, în special, datorită vastei sale răspândiri determinate de apariția și dezvoltarea spectaculoasă a calculatoarelor personale (atât de familiare astăzi).

Lucrarea nu urmărește prezentarea în mod special a unei anume variante BASIC (o abordare vastă și complexă a diverselor familii și generații BASIC fiind prezentată în [4]), ci utilizarea acestuia ca instrument în soluționarea problemelor. Din aceasta cauză ne-am oprit în mod special asupra elementelor fundamentale ale limbajului, de regulă, universal valabile în familiile și generațiile sale, referirea la o anume versiune făcându-se numai când am considerat a fi absolut necesară și cu specificarea acesteia.

Prin modul de abordare, prin natura problemelor propuse și soluționate, prin cele peste 30 de exemple și 40 de aplicații (însoțite de rezultatele execuțiilor efectuate cu date de test) prezentate în capitolul 5, lucrarea are un profund caracter practic-aplicativ .

Lucrarea se adresează, în special, elevilor din ciclurile gimnazial și liceal, profesorilor care predau noțiuni introductive de programare, precum și studenților și tuturilor acestora care au în propria casă sau la îndemâna un "home-computer" și vor să-l utilizeze.

Autorii

Lucrări în domeniu, care să includă în cuprinsul lor - total sau parțial - asemenea aspecte s-au scris foarte multe, dar adesea îndreptate spre specialiști. Ceea ce aduce nou această carte este abordarea conținutului său într-un mod pedagogic, pe scări graduate de complexitate și informație, într-o viziune integratoare.

Plecând de la cunoașterea unei probleme - în general - lucrarea trece prin toate etapele logice ale rezolvării ei - respectiv: localizarea descoperirea pe post de concepte algoritmicul descrierea necesită - ca apoi să facă introucherea în sfera informatică și rezolvării ei, respectiv alcătuirea programului. Prin parcurgerea acestor etape se creează formarea și dezvoltarea unei modalități de abordare și gândire logică nu numai în rezolvarea problemelor de matematică, dar și în

CUPRINS

1. NOȚIUNI GENERALE	1
1.1 Scurt istoric	1
1.2 Ce este un calculator numeric?	3
● Hardware	4
● Software	6
1.3 Cum rezolvăm o problemă?	7
● Formularea problemei	7
● Fundamentarea teoretică	7
● O primă concluzie	9
● Algoritmul (pașii) rezolvării problemei	9
1.4 Cîteva concluzii	10
2. ALGORITMI. METODE DE REPREZENTARE	11
2.1 Despre algoritmi	11
● Definiție	11
● Exemple clasice de algoritmi	11
● Conexiunile algoritmilor cu mulțimile valorilor variabilelor de intrare și de ieșire	14
● Proprietățile algoritmilor	16
2.2 Descrierea algoritmilor	17
● Schemele logice	17
● Limbajul algoritmic 'pseudocod'	19
2.3 Structuri fundamentale	21
● Structuri liniare	22
● Structuri alternative	23
● Structuri repetitive	25
2.4 Utilizarea variabilelor indexate	30
3. ELEMENTE ALE LIMBAJULUI BASIC	32
3.1 Cîteva precizări	32

3.2	Moduri de operare și comenzile interpretorului	34
●	Inițializarea interpretorului	34
●	Moduri de operare	35
●	Salvări, restaurări	36
●	Ștergeri, modificări, listări	37
●	Execuții	39
●	Reluări, trasări	40
●	Revenirea în sistemul de operare	40
3.3	Propoziții și fraze (programe) BASIC	40
3.4	Vocabular și elemente de gramatică	43
●	Setul de caractere	43
●	Constante	43
●	Cuvinte rezervate	43
●	Identificatori	44
●	Variabile	44
●	Operatori	46
●	Expresii	47
3.5	Instrucțiuni de intrare / ieșire	48
3.6	Operații aritmetice și logice	54
●	Operații aritmetice	54
●	Operații logice	56
●	Operații cu variabile șir	58
3.7	Funcții și subrutine	58
●	Funcții predefinite	59
●	Funcții definite de utilizator	63
●	Subrutine BASIC	64
●	Subrutine BETA BASIC	67
3.8	Instrucțiuni de control	68
●	Instrucțiuni alternative	68
●	Instrucțiuni repetitive	70
●	Instrucțiuni de salt	72
3.9	Lucrul cu fișiere	76
●	Deschiderea și închiderea fișierelor	76
●	Fișiere secvențiale	77
●	Lucrul cu fișiere în acces direct	79
3.10	Înlănțuirea programelor	83
3.11	Elemente de grafică	84
●	Fereastra, vizor, scalare, decupare	85
●	Instrucțiuni grafice	87

4. SPRE O PROGRAMARE MODULARĂ ȘI STRUCTURATĂ	92
5. APLICAȚII	105
5.1 Numere întregi, baze de numerație	106
● Împărțire cu rest a două numere naturale	106
● Împărțirea cu rest a două numere întregi	107
● Determinarea CMMDC al două numere întregi	108
● Determinarea CMMMC al două numere întregi	110
● Determinarea CMMDC al maxim 20 de numere întregi	111
● Determinarea CMMMC al maxim 20 de numere întregi	113
● Generarea de numere prime	115
● Determinarea divizorilor unui număr întreg	118
● Conversia unui număr natural din baza 10 într-o bază mai mică decât 10 ..	119
● Conversia unui număr natural din baza 10 în toate bazele mai mici decât 10	121
● Conversia unui număr binar în baza 10	123
● Descompunerea unui număr întreg în factori primi	125
5.2 Vectori	128
● Determinarea elementului minim / maxim și al locului său într-un vector .	128
● Suma elementelor unui vector	130
● Căutarea unei element într-un vector	131
● Determinarea numerelor elementelor pozitive, nule și negative conținute într-un vector	132
● Interschimbarea a două elemente ale unui vector	134
● Inversarea ordinii elementelor unui vector	135
● Interschimbarea a k elemente de la începutul unui vector cu cele k elemente de la sfârșitul său	136
● Ordonarea crescătoare / descrescătoare a elementelor unui vector	137
● Rotirea stînga / dreapta a elementelor unui vector	139
● Deplasarea stînga / dreapta a elementelor unui vector	141
5.3 Mulțimi	143
● Validarea unei mulțimi de numere	143
● Reuniunea a două mulțimi	144
● Intersecția a două mulțimi	147
● Diferența a două mulțimi	148
● Produsul cartezian al două mulțimi de numere	150
5.4 Matrici, determinanți	151
● Citirea și listarea unei matrice	151
● Suma elementelor unei matrice	152
● Suma a două matrici	153
● Produsul a două matrici	154

92	● Transpusa unei matrice	155
	● Calculul determinantului unei matrice	156
102	● Inversarea unei matrici pătrate	159
106	5.5 Șiruri de caractere	161
106	● Conversia unui număr natural ($32 \leq N \leq 26$) în caracter	161
107	● Crearea unui cap de tabel	162
108	● Căutarea unui subșir într-un șir dat	163
110	5.6 Fișiere	164
111	● Actualizarea fișierului secvențial CLASA	165
113	● Actualizarea fișierului cu acces direct CLASAD	169
112		
ANEXA A		173
ANEXA B		177
121		
Bibliografie		183

1. NOȚIUNI GENERALE

1.1 Scurt istoric

De-a lungul existenței sale omul a căutat permanent soluții pentru a se elibera de efortul fizic, de muncile oboseitoare sau plictisitoare. Diversificarea continuă a activităților sale a determinat și determină și acum necesitatea găsirii unor soluții noi la vechile și permanentele sale probleme, ceea ce a dus, și duce, la perfecționarea continuă a instrumentelor cu care operează.

În confruntarea continuă cu natura și cu timpul s-a născut de mult ideea inventării unor mașini automate de calculat și, de ce nu, a unor mașini automate care să-l ajute în gândire.

Deși idei și încercări în acest sens datează de prin secolul al XVII-lea (Napier-1617, Pascal-1642) și mai cu seamă din secolul al XIX-lea (Babbage lansează în 1820 ideea realizării unei mașini de calculat care, pe lângă efectuarea de operații aritmetice să aibă posibilitatea de reținere a comenzilor necesare soluționării problemei, precum și a rezultatelor, pe care să le furnizeze ulterior), realizarea practică a primelor calculatoare este posibilă abia către jumătatea secolului al XX-lea (descoperirea de către Lee de Forest în anul 1906 a triodei a determinat o puternică dezvoltare a electronicii, bazate pe tuburi electronice, în prima jumătate a acestui secol).

În anul 1945 apare ENIAC (Electronic Numerical Integrator And Calculator) care este considerat a fi primul calculator electronic. Datorită celor circa 20000 de tuburi electronice care îl compuneau, dimensiunile sale erau considerabile.

Apariția calculatoarelor electronice a produs o puternică revoluție în modul de abordare și rezolvare a problemelor, determinând practic apariția și dezvoltarea informaticii - ca știință a prelucrării automate a datelor - cu puternice implicații asupra tuturor domeniilor activității umane.

Concomitent cu evoluțiile tehnologice în domeniul dispozitivelor și circuitelor electronice, calculatorul a evoluat continuu, stăbătind mai multe generații (tabelul 1.1).

Prima generație, cuprinzând calculatoarele electronice din perioada 1946-1956, se caracterizează în principal prin utilizarea ca elemente de bază a tuburilor electronice, fiabilitate scăzută, memorii interne mici, viteze de calcul reduse și o programare greoaie (în limbaj de asamblare). Datorită dimensiunilor și dificultăților în exploatare au cunoscut o utilizare restrânsă.

Generația a doua, cuprinzând calculatoarele electronice realizate între anii 1957-1963 își datorează existența dezvoltării și utilizării tehnologiilor bazate pe tranzistori, ceea ce a determinat o creștere a fiabilității și vitezei de calcul concomitent cu reducerea dimensiunilor.

Apar memoriile bazate pe celule din ferită. Se pun la punct metode de gestiune a resurselor calculatorului; apare necesar, și este creat, un sistem de programe care să dirijeze funcționarea optimă a acestuia (sistemul de operare) astfel că începând cu generația a doua, sistemele de calcul se compun din două părți mari: **hardware** (dispozitivele și echipamentele componente ale unui sistem) și **software** (totalitatea programelor cu care este echipat sistemul de calcul). De asemenea apar și se

1. Noțiuni generale

dezvoltă limbajele de programare de nivel înalt (universale), relativ independente de calculator (FORTRAN, COBOL etc.). Aria de utilizare a calculatoarelor se mărește.

GENERAȚII DE CALCULATOARE ELECTRONICE [17]

Generație	Perioadă	Principala tehnologie hardware	Alte tehnologii hardware	Limbaje	Alte proprietăți ¹⁾
I	1964-1956	Tuburi electronice	Tambur magnetic	De asamblare	MI-2 KO V-10 KIPS
II	1957-1963	Tranzistori	Memorii din ferite	De nivel înalt: FORTRAN, COBOL	MI-32 KO V-200 KIPS
III	1964-1981	Circuite integrate	Memorii semiconductoare, discuri magnetice	De nivel foarte înalt: PASCAL, LISP, limbaje grafice etc.	MI-2 MO V-5 MIPS
IV	1982-1989	Circuite integrate pe scară largă și foarte largă	Memorii cu bule magnetice, discuri optice	ADA, limbaje orientate obiect	MI-8 MO V-30 MIPS Supercalculatoare
V	după 1989	Circuite integrate pe scară extrem de largă	Arhitecturi paralele	Limbaje concurente, LIMBAJ NATURAL, programare funcțională	Inteligență artificială MI>16 MO V>1 GIPS Vedere artificială, tehnologia vorbirii

¹⁾ MI (memorie internă) 1 MO = 1000 KO

V (viteza de calcul) 1 GIPS = 1000 MIPS = 1000000 KIPS (kiloinstrucțiuni pe secundă)

Tabelul 1.1

Generația a treia (1964-1981) este impusă de apariția circuitelor integrate. Dimensiunile calculatoarelor se reduc considerabil, crește substanțial dimensiunea memoriei interne (bazată pe semiconductoare), crește viteza de prelucrare, apare discul magnetic ca suport extern de informații de înaltă eficiență. Apar limbajele de nivel foarte înalt și limbajele grafice. Apare microprocesorul pe 8 biți.

Perfecționarea tehnologiilor, miniaturizarea și reducerea considerabilă a prețului de cost, au făcut posibilă punerea în practică a dezideratului "un calculator pentru fiecare", astfel încât, anul 1972 (anul realizării microprocesorului 8080) marchează începerea producției unor noi tipuri de calculatoare "calculatoarele personale" a căror primă generație (tabelul 1.2) se va dezvolta pînă către anul 1980.

Generația a patra de calculatoare (1982-1989) se caracterizează prin utilizarea circuitelor integrate pe scară largă și foarte largă precum și a microprocesoarelor. Se înregistrează salturi spectaculoase în creșterea capacității de stocare pe diferite suporturi externe de informații. Prețurile de cost scad considerabil, comparativ cu creșterea performanțelor sistemelor de calcul.

Apariția în 1980 a microprocesorului pe 16 biți 8086 determină, în paralel, lansarea generației

a doua de microcalculatoare personale al cărei produs de referință este calculatorul IBM-PC. În 1986 este pus la punct microprocesorul 80386. Îi urmează, la scurt timp, microprocesorul 80486 (32 de biți). Acestea marchează trecerea la a treia generație de microcalculatoare personale.

GENERAȚII DE CALCULATOARE PERSONALE

Generație	Perioada	Microprocesor	Alte caracteristici
I	1972-1980	8 biți: 8080, Z-80, 8085, MOS6502, etc.	MI-64 KO Sistem de operare CP/M
II	1981-1986	16 biți: 8086, 8088, 80286, etc.	MI-640-1000 KO Hard disk de zeci de MO, facilități grafice, sistem de operare MS-DOS
III	după 1986	16 și 32 biți: 80386, 80486, etc.	MI>1 MO, hard disk de sute de MO, facilități grafice deosebite, posibilități de lucru în rețea

Tabelul 1.2

După numai circa 45 de ani de evoluție, transformările suferite de calculatorul electronic din punct de vedere al realizării tehnologice sînt nu numai fundamentale dar și spectaculoase astfel că, astăzi, calculatorul a devenit prezent practic, în toate domeniile de activitate socială sau personală, a ajuns un instrument de lucru din ce în ce mai necesar.

Care ar fi perspectivele? Încă din 1981 Japonia lansează proiectul calculatorului de generația a cincea. Această generație (a cărei apariție se întrevide a avea loc după 1990) ar urma să se deosebească de generațiile anterioare (în speță de generația a patra) în principal, nu atît printr-o nouă tehnologie de construcție (hardware), ci printr-o nouă concepție software, aceea a inteligenței artificiale.

În concepția specialiștilor japonezi calculatoarele din această generație vor deveni sisteme de procesare a informației de cunoaștere. Se presupune că această generație va definitiva orientarea informațională a societății umane cu implicații majore în evoluția acesteia.

1.2 Ce este un calculator numeric?

După cum îl înfățișează și numele, în general, *un calculator este un sistem fizic care prelucrează datele într-o formă prestabilită și furnizează rezultatele într-o formă accesibilă utilizatorului sau altui dispozitiv cu care este conectat.*

Din mulțimea de tipuri de calculatoare, calculatoarele numerice sînt acelea care prelucrează informații codificate numeric efectuînd operații aritmetice, funcții logice și transferuri de informații.

Așa cum am văzut, începînd cu generația a doua, calculatorul electronic este alcătuit din două componente de bază:

- hardware (totalitatea dispozitivelor și echipamentelor fizice) calculatorul propriu-zis;
- software (totalitatea programelor cu care este "echipat" acesta).

● **Hardware**

Din punct de vedere hardware un calculator electronic numeric trebuie să dispună în principal de următoarele componente:

- memorie internă (MI);
- unitate centrală (UC);
- dispozitive periferice (P):
 - de introducere a datelor;
 - de afișare a rezultatelor;
 - eventual, alte dispozitive periferice.

Cum ne interesează în special microcalculatoarele, ne oprim puțin asupra principalelor componente ale acestora (figura 1.1).

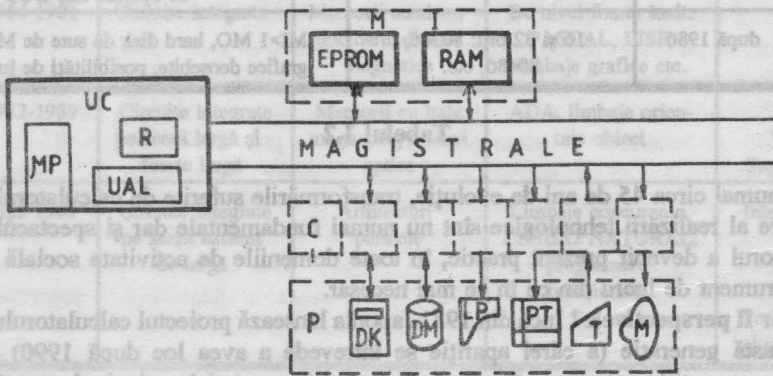


Fig. 1.1

Memoria internă (MI) este un dispozitiv alcătuit din 'locații' succesive adresabile, avînd rolul de a memora principalele programe care asigură gestiunea resurselor calculatorului, a programului utilizatorului, precum și a datelor cu care se lucrează. În general, memoria internă este împărțită în două părți distincte: memoria EPROM (conține informații privitoare la diferitele caracteristici fizice ale mașinii și programul "încărcător", program ce preia controlul la alimentarea cu tensiune a calculatorului), avînd proprietatea de a-și păstra conținutul și după decuplarea calculatorului, dar care permite numai citirea informațiilor stocate, și memoria RAM care permite operații de citire / scriere și în care sînt reținute temporar programe și date. Memoria RAM își pierde conținutul la decuplarea alimentării calculatorului.

Unitatea centrală (UC) este un dispozitiv care asigură dirijarea funcționării normale a calculatorului în ansamblu cît și a principalelor sale componente. Unitatea centrală ia decizii privind modul și ordinea de execuție a diferitelor operații în interiorul calculatorului și semnalează apariția unor condiții de eroare sau funcționarea anormală a acestuia.

Componenta fundamentală a unității centrale o constituie microprocesorul (μP), el fiind de fapt elementul care, prin sistemul de întreruperi, coordonează funcționarea tuturor componentelor calculatorului. După cum am văzut din tabelul 1.2, microprocesorul, prin caracteristicile sale, este un

element definitor al generațiilor de microcalculatoare și, implicit, al performanțelor unui calculator. O importanță aparte, prin rolul său în funcționarea unității centrale, o are unitatea aritmetică și logică (UAL), unitate ce asigură execuția efectivă a operațiilor aritmetice și logice.

De asemenea, unitatea centrală, mai conține o memorie proprie (registre - R) precum și o serie de alte circuite menite să asigure sau să îmbunătățească performanțele calculatorului.

Pentru a executa o instrucțiune (de exemplu adunarea a două numere $C = A + B$) unitatea centrală execută următoarele operații:

- preia din memoria internă instrucțiunea ($C = A+B$) și o decodifică;
- calculează adresele operanzilor, îi identifică (A,B) și efectuează deplasările necesare (aduce operandul A în memoria proprie);
- cu ajutorul UAL efectuează operația ($A + B$) cu păstrarea rezultatului;
- depune rezultatul în memoria internă la locația corespunzătoare (C);
- semnalează sfârșitul și corectitudinea executării operației.

Referitor la funcționarea unității centrale trebuie subliniat faptul că ea execută operațiile pas cu pas (pe rînd) la un moment dat executînd o singură operație.

Dispozitivele periferice de introducere a datelor sînt multiple. În cazul calculatoarelor personale acest dispozitiv îl constituie, de obicei, tastatura (T).

Dispozitivele de afișare a datelor alcătuiesc o categorie complexă și variată de dispozitive. Din aceasta fac parte, în primul rînd monitorul (M) - banalul televizor - care poate fi color sau monocrom, dispozitiv utilizat și pentru conversația cu calculatorul, apoi imprimanta (alfanumerică sau grafică), plotterul, etc.

Pe lângă aceste dispozitive periferice, o importanță deosebită o au suporturile de memorie externă, fără de care, în condițiile actuale este greu de conceput utilizarea eficientă a unui calculator. Prețurile relativ scăzute în ultima vreme ale acestor componente le fac tot mai accesibile utilizatorului neprofesionist, avantajele utilizării lor fiind evidente. În această categorie amintim numai, benzile magnetice (BM), discurile magnetice (DM) (cu capacități de stocare de ordinul sutelor de megaocteți), precum și familiarele dischete (DK) (și ele cu capacități de ordinul megaocteților).

Atît memoria internă cît și dispozitivele periferice sînt conectate la unitatea centrală prin magistrale - prin care se asigură atît transferul datelor și instrucțiunilor cît și controlul funcționării acestora. Conectarea dispozitivelor periferice la magistrale este efectuată cu ajutorul unor dispozitive (cuploare, interfețe) specifice fiecăruia.

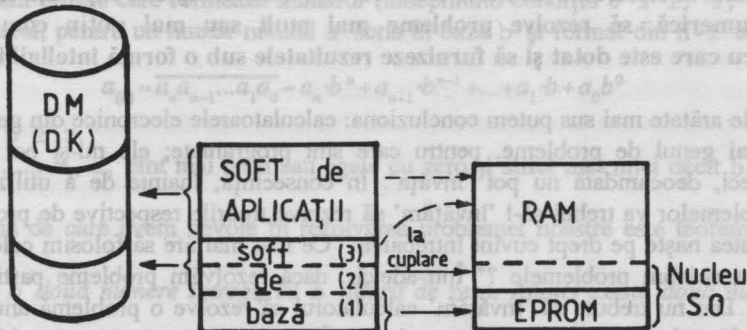


Fig. 1.2

● Software

Cea de a doua componentă de bază a unui calculator numeric este cunoscută sub numele de software și, la rândul său, este constituită din două părți mari (figura 1.2):

- software-ul de bază;
- software-ul de aplicații.

Software-ul de bază este alcătuit din totalitatea programelor fără de care utilizarea calculatorului nu este posibilă. O parte din software-ul de bază (1) este depus în memoria EPROM. La cuplarea calculatorului în memoria internă sînt încărcate automat de pe disc și alte programe absolut necesare dirijării funcționării calculatorului (2), programe care împreună cu cele aflate în memoria EPROM alcătuiesc 'nucleul sistemului de operare', nucleu existent în memoria internă a calculatorului pe toată durata funcționării lui. Restul componentelor (3) sînt încărcate în memorie numai dacă sînt solicitate de sistem sau utilizator și numai pe durata utilizării lor (din această categorie fac parte compilatoarele, diferitele utilitare etc.).

Observație. În cazul microcalculatoarelor care nu dispun de unități de disc toate componentele software legate de funcționarea calculatorului precum și un minim pentru exploatarea acestuia se află permanent în memoria EPROM. În cazul acestor calculatoare nu putem vorbi de "sistem de operare" în înțelesul deplin al cuvîntului.

Software-ul de aplicații este constituit din mulțimea programelor destinate rezolvării diferitelor tipuri de probleme.

În această categorie intră și programele create de utilizator; aceste programe sînt lansate în execuție, de regulă de utilizator și sînt prezente în memorie numai pe durata execuției.

Observație. În cazul microcalculatoarelor care nu dispun de discuri magnetice (fixe sau dischete) pentru încărcarea/salvarea programelor este necesară existența cel puțin a unei unități de bandă magnetică (de regulă casetofon), comanda pentru efectuarea operației (salvare/restaurare) fiind dată de utilizator; în caz contrar programul 'se pierde' la decuplarea calculatorului.

Calculatorul electronic numeric este deci o mașină complexă capabilă să primească date sub formă numerică, să rezolve probleme mai mult sau mai puțin complicate conform programelor cu care este dotat și să furnizeze rezultatele sub o formă inteligibilă utilizatorului.

Din cele arătate mai sus putem concluziona: calculatoarele electronice din generațiile I-IV pot soluționa numai genul de probleme, pentru care sînt programate; ele nu-și pot construi singure algoritmi și deci, deocamdată nu pot 'învăța'. În consecință, înainte de a utiliza calculatorul la rezolvarea problemelor va trebui să-l 'învățăm' să rezolve tipurile respective de probleme.

S-ar putea naște pe drept cuvînt întrebarea: "Ce rost mai are să folosim calculatorul dacă tot noi trebuie să rezolvăm problemele?" Într-adevăr, dacă rezolvăm probleme particulare întrebarea este justificată. Dar nu trebuie să 'învățăm' calculatorul să rezolve o problemă anume ci, un tip de probleme (să-i dăm algoritmul general de rezolvare). În acest caz, pentru toate problemele de acel tip dîndu-i numai datele specifice (date de intrare) calculatorul va determina, urmărind algoritmul general,

răspunsul căutat. De data aceasta utilizarea calculatorului este eficientă deoarece soluțiile problemei se obțin fără a mai participa efectiv la efectuarea calculelor.

Calculatorul este deci, o mașină capabilă să rezolve orice problemă cu condiția să fi fost 'învățat' cum să o facă.

În consecință, înainte de toate, cum rezolvăm o problemă?

1.3 Cum rezolvăm o problemă?

În general problemele pe care dorim să le rezolvăm cu calculatorul sînt teoretic cunoscute dar ele necesită fie un volum mare de calcule, fie operații stereotipe, plictisitoare.

În aceste cazuri comod este să introducem datele problemei în calculator și să obținem rezultate. Am văzut însă că, pentru a putea determina soluțiile unei probleme cu ajutorul calculatorului, este necesar ca acesta 'să cunoască' pas cu pas întreaga succesiune de operații ce trebuie efectuate pentru rezolvarea acesteia.

Să rezolvăm o problemă simplă (de nivelul clasei a V-a): trecerea unui număr natural din baza 10 într-o bază b , cu b mai mare sau egal cu 2 și mai mic decît 10.

Tocmai pentru că este o problemă banală să vedem cum o rezolvăm sau, mai bine zis, cum o pregătim pentru a putea fi rezolvată pe calculator.

● Formularea problemei

Să se treacă un număr natural a din baza 10 în baza b ($b \in \mathbb{N}$ și $2 \leq b < 10$)

● Fundamentarea teoretică

Un număr natural, de exemplu 157, spunem că este scris în baza 10 dacă:

$$157_{(10)} = 1 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0$$

unde 1, 5 și 7 sînt cifrele care formează numărul (îndeplinind condiția $0 \leq 1, 5, 7 \leq 9$) iar 10 este baza. În general, pentru un număr natural a scris în baza b și format din $n+1$ cifre avem:

$$a_{(b)} = \overline{a_n a_{n-1} \dots a_1 a_0} = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \dots + a_1 \cdot b + a_0 b^0$$

unde: $b \in \mathbb{N}$ și $b \geq 2$,

iar cifrele $a_n, a_{n-1}, \dots, a_1, a_0$ sînt mai mari sau egale cu zero și strict mai mici decît b .

O teoremă de care avem nevoie în rezolvarea problemei noastre este teorema împărțirii cu rest:

Fie a și b două numere întregi, cu b diferit de zero. Atunci există două numere întregi q și r , astfel încît: $a = bq + r$ și $0 \leq r < |b|$.

Numerele q și r determinate în aceste condiții sînt unice.

Cu alte cuvinte, împărțind un număr întreg a (deîmpărțitul) la un număr întreg b (împărțitor) obținem un cât q și un rest r . Restul este totdeauna mai mic decât împărțitorul, luat în valoare absolută, și mai mare sau egal cu 0. Dacă r (restul) este egal cu zero, atunci spunem că b divide a sau că b este divisor al lui a .

Exemplul 1.1

Fie $a = 125_{(10)}$; să se determine numărul $c_{(2)} = a_{(10)}$.

Aplicând teorema împărțirii cu rest și parcurgând schema din figura 1.3 rezultă:

$$125_{(10)} = 1111101_{(2)}$$

Verificare: utilizând modul de scriere al numerelor în baza 2 rezultă:

$$1111101_{(2)} - 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 125_{(10)}$$

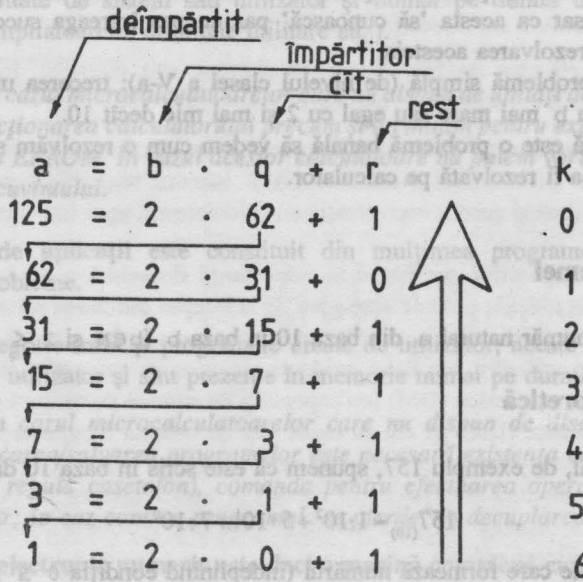


Fig. 1.3

Exemplul 1.2

Să convertim în baza 8 numărul $342_{(10)}$. Folosind schema din figura 1.4 rezultă:

$$342_{(10)} = 526_{(8)}$$

Verificare:

$$526_{(8)} - 5 \cdot 8^2 + 2 \cdot 8^1 + 6 \cdot 8^0$$

a	b	q	r	k	
342	=	8 ·	42	+ 6	0
↓					↑
42	=	8 ·	5	+ 2	1
↓					↑
5	=	8 ·	0	+ 5	2

Fig.1.4

Privind comparativ cele două exemple observăm că am folosit aceeași metodă, și anume, am împărțit (utilizând teorema împărțirii cu rest) numărul dat (125 respectiv 342) scris în baza zece la noua bază (2 respectiv 8) obținând un cît și un rest. Am reținut restul iar cîțul a devenit deîmpărțit și am efectuat o nouă împărțire. Pînă cînd? Pînă cînd cîțul a ajuns mai mic decît împărțitorul și deci, în ultima împărțire, a devenit el însuși rest. Numărul căutat se formează din cifrele rezultate ca rest, luate în ordine inversă obținerii lor.

Examinați ultimul paragraf. Ce constatați ?

● O primă concluzie

Modul de rezolvare nu depinde nici de numărul dat, nici de baza aleasă (respectînd însă condițiile impuse). Am alcătuit deci o metodă generală care, transpusă pe calculator, ne permite rezolvarea problemei pentru orice pereche de numere naturale (a, b) cu b mai mare sau egal cu 2 și strict mai mic decît 10.

După cum am văzut însă în capitolul 1.2 deocamdată calculatorul rezolvă problemele numai 'pas cu pas'. Care ar fi pașii rezolvării problemei propuse ?

● Algoritmul (pașii) rezolvării problemei

Înainte de a trece efectiv la scrierea pașilor rezolvării problemei, să facem o precizare: pentru a putea ști cîte cifre are numărul a scris în baza b va trebui să numărăm de cîte ori efectuăm împărțirea cu rest; pentru aceasta vom folosi un contor k cu valoare inițială egală cu zero în care, la fiecare trecere (de fiecare dată cînd executăm împărțirea), adunăm o unitate.

Pașii rezolvării problemei ar putea fi:

- pasul 1: se stabilesc valorile numerelor a și b ;
- pasul 2: se verifică dacă sînt îndeplinite condițiile impuse: $(a, b) \in \mathbb{N}$ și $2 \leq b \leq 9$);
 - dacă DA continuăm cu pasul 3;
 - dacă NU cel puțin unul din numerele date este eronat ales și rezolvarea problemei se întrerupe (se sare la pasul 9 sau, eventual, la pasul 1 pentru a corecta eroarea);

1. Noțiuni generale

- pasul 3: se atribuie contorului k valoarea 0;
- pasul 4: se efectuează împărțirea cu rest:
$$a = b q + r_k$$
- pasul 5: se adună o unitate la valoarea contorului k ;
- pasul 6: cîțul devine deîmpărțit $a = q$ (vechea valoare a lui a nu ne mai interesează), iar restul r_k se reține;
- pasul 7: se verifică dacă $a < b$;
 - dacă DA se continuă cu pasul 8;
 - dacă NU se revine la pasul 4;
- pasul 8: se scrie rezultatul sub forma:

$$\overline{r_k r_{k-1} \dots r_1 r_0}$$

- pasul 9: stop.

După cum vedem modul de pregătire a rezolvării unei probleme pentru a putea fi transpusă pe calculator diferă întrucîtva de modul în care se rezolvă în general problemele. Aceasta se datorează caracterului de generalitate pe care trebuie să-l aibă metoda de rezolvare pe care dorim să o utilizăm.

1.4 Cîteva concluzii

Cel puțin pînă în prezent calculatoarele pe care le avem la îndemînă nu sînt dotate cu inteligența de a învăța. Ele sînt niște mașini care pot soluționa rapid și corect o gamă foarte largă de probleme cu condiția să fi fost învățate să o facă.

Transpunerea pe calculator a modului de rezolvare a unei probleme presupune ca primă condiție cunoașterea teoretică profundă a problemei și determinarea riguroasă a unei succesiuni de operații (pași) care (în condițiile în care se îndeplinesc restricțiile precizate) să ducă în mod univoc la determinarea soluției (soluțiilor) acesteia.

Calculatorul execută automat (pas cu pas) operațiile precizate fără a participa în vreun fel la corectarea eventuală a acestora sau la soluționarea situațiilor ambigue. Din această cauză în precizarea succesiunii operațiilor pentru rezolvarea unei probleme trebuie analizate și precizate operațiile de executat în toate cazurile ce pot apare.

Transpunerea pe calculator a modurilor de rezolvare ale unor tipuri de probleme nu este o treabă prea simplă dar nici foarte complicată. Ea presupune, însă, pe lîngă cunoașterea teoretică aprofundată a problemei (problemelor) și unele cunoștințe privind întocmirea și reprezentarea algoritmilor, precum și cel puțin a unui limbaj de programare.

2. ALGORITMI. METODE DE REPREZENTARE

2.1 Despre algoritmi

● Definiție

În rezolvarea diferitelor probleme (nu numai de matematică), funcție de complexitatea acestora, trebuie, de obicei, ca pentru a obține soluția căutată să executăm, pornind de la anumite date, o succesiune de operații aritmetice și/sau logice. Știm de asemenea din experiență că aceste operații nu se fac la întâmplare, ci într-o ordine determinată de natura problemei și de unele rezultate intermediare obținute. Pentru a putea executa această succesiune de operații în mod automat (cu ajutorul calculatorului), deci fără aportul creator al omului, trebuie (așa cum am văzut în capitolul 1.3) ca ele să fie valabile pentru toate problemele de același tip, astfel încât schimbând numai datele inițiale, soluțiile să fie determinate în mod automat.

De asemenea, am mai văzut că fiecare operație trebuie să aibă un mod cunoscut de efectuare, numărul lor trebuie să fie finit, iar parcurgerea întregii succesiuni să se încadreze într-un interval de timp rezonabil.

Odată stabilite operațiile și ordinea în care urmează a fi executate (în condițiile precizate mai sus) spunem că am alcătuit un algoritm de rezolvare a problemei respective.

Deci, am putea concluziona că, un algoritm este o mulțime finită de operații cunoscute care se execută într-o ordine stabilită, astfel încât, pornind de la un set de date (datele problemei) ce îndeplinesc anumite condiții, obținem, într-un interval de timp finit, un set de valori (soluțiile problemei).

● Exemple clasice de algoritmi

Să ne oprim asupra a doi algoritmi îndeobște cunoscuți: *algoritmul lui Euclid* de determinare a celui mai mare divizor comun a două numere întregi și *algoritmul lui Eratostene* de determinare a numerelor prime cuprinse în intervalul $[2, n]$, unde n este un număr natural mai mare sau egal cu doi, algoritm cunoscut sub numele de *ciurul lui Eratostene*.

Exemplul 2.1.

Algoritmul lui Euclid: pentru a obține c.m.m.d.c. a două numere întregi a și b cu b diferit de zero împărțim pe a cu b ; dacă restul împărțirii (r_1) este zero atunci b este c.m.m.d.c.; dacă nu, împărțim pe b cu restul împărțirii anterioare (r_1) și obținem restul r_2 ; apoi împărțim pe r_1 cu r_2 și obținem r_3 , și așa mai departe. Ultimul rest nenul este c.m.m.d.c. al numerelor date (a și b).

Înainte de a explicita algoritmul se impun două precizări:

1. Deoarece dacă d este c.m.m.d.c. al numerelor a și b atunci și $-d$ este c.m.m.d.c. al lui a și b și cum c.m.m.d.c. al numerelor a și b nu depinde nici de ordinea, nici de semnul lor, atunci, pentru a afla c.m.m.d.c. este suficient să aplicăm algoritmul lui Euclid pentru $|a|$ și $|b|$, adică pentru două numere naturale din care cel puțin unul trebuie să fie diferit de zero.

2. Din motive de generalitate și pentru a nu genera confuzii în algoritm se vor utiliza două variabile $d = \max(|a|, |b|)$ și $i = \min(|a|, |b|)$ reprezentând deîmpărțitul, respectiv împărțitorul cunoscuți de la împărțirea cu rest. În acest caz c.m.m.d.c. al numerelor a și b va fi numărul i obținut la prima împărțire cu rest zero.

Transpus pe pași (operații), algoritmul este:

- pasul 1: se aleg numerele întregi a și b ;
- pasul 2: $|a| > |b|$?
 - dacă DA, atunci $d = |a|$ și $i = |b|$;
 - dacă NU, atunci $d = |b|$ și $i = |a|$;
- pasul 3: $i > 0$?
 - dacă DA, atunci se continuă cu pasul 4;
 - dacă NU, atunci algoritmul nu se poate aplica și se sare la pasul 10; (în acest punct putem alcătui algoritmul și în altă variantă, cu salt la pasul 1 pentru o eventuală reluare cu alte valori atribuite variabilelor a și b)
- pasul 4: se efectuează împărțirea:

$$d = i \cdot q + r;$$
- pasul 5: $r = 0$?
 - dacă DA, atunci se sare la pasul 9;
 - dacă NU, atunci se continuă cu pasul 6;
- pasul 6: $d = i$ (împărțitorul devine deîmpărțit);
- pasul 7: $i = r$ (restul devine împărțitor);
- pasul 8: se sare la pasul 4;
- pasul 9: i este c.m.m.d.c. al lui a și b ;
- pasul 10: stop.

Verificăm cu un exemplu numeric:

- pasul 1: $a = 21$ $b = 14$;
- pasul 2: $|21| > |14|$?
 - DA => $d = 21$ $i = 14$;
- pasul 3: $i > 0$ ($14 > 0$)?
 - DA => se trece la pasul 4;
- pasul 4: $d = i \cdot q + r$ ($21 = 14 \cdot 1 + 7$) => $r = 7$;
- pasul 5: $r = 0$?
 - NU => se trece la pasul 6;
- pasul 6: $d = 14$;
- pasul 7: $i = 7$;
- pasul 8: se sare la pasul 4;
- pasul 4: $d = i \cdot q + r$ ($14 = 7 \cdot 2 + 0$) => $r = 0$;

- pasul 5: $r = 0$?
- DA => se sare la pasul 9;
- pasul 9: $7 = (21, 14)$ 7 este c.m.m.d.c. al numerelor 21 și 14
- pasul 10: stop.

Rezultatul este corect. Înlocuind pe a și b cu alte valori (respectând însă, bineînțeles, condițiile impuse) se vor obține soluții corecte. Deci transpunerea pe pași a algoritmului a fost efectuată corect.

Exemplul 2.2.

Ciurul lui Eratostene permite, așa cum am arătat obținerea tuturor numerelor prime mai mici sau egale cu un număr natural $n \geq 2$ dat. Metoda constă în următoarele: fiind dat șirul numerelor naturale mai mari sau egale cu 2 și mai mici sau egale cu n, se pornește de la primul număr din șir (numărul 2) care este număr prim și se înlătură din șir toți multiplii acestuia mai mici sau egali cu n; se trece apoi la următorul număr rămas în șir (numărul 3) și se înlătură toți multiplii acestuia care mai sînt în șir, și așa mai departe pînă la cel mai mare număr prim mai mic sau egal cu \sqrt{n} . Numerele rămase în șir constituie mulțimea numerelor prime căutate.

Fie p_k numărul prim căruia îi determinăm la cea de-a k parcurgere multiplii; să schițăm pașii algoritmului :

- pasul 1: se alege numărul n ($n \in \mathbb{N}$ și $n \geq 2$);
- pasul 2: se generează șirul numerelor naturale din intervalul $[2, n]$;
- pasul 3: $k = 1$ (sîntem la primul număr prim);
- pasul 4: se atribuie lui p_k valoarea celui de al k-lea număr din șir;
- pasul 5: se verifică dacă $p < \sqrt{n}$
 - dacă DA, atunci se trece la pasul 6;
 - dacă NU, atunci se trece la pasul 9;
- pasul 6: se determină toți multiplii lui p_k mai mici sau egali cu n și se înlătură din șir;
- pasul 7: se adună 1 la valoarea curentă a lui k;
- pasul 8: se sare la pasul 4;
- pasul 9: se afișează numerele rămase în șir; acestea formează mulțimea numerelor prime căutate;
- pasul 10: stop.

Verificăm cele arătate pentru $n = 26$:

- pasul 1: $n = 26$;
- pasul 2: șirul este:

2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	

- pasul 3: $k = 1$;
- pasul 4: $p_k = 2$;
- pasul 5: $p_k \leq \sqrt{n}$ ($2 \leq \sqrt{26}$) ?
- DA => se trece la pasul 6;

2. Algoritmi. Metode de reprezentare

- pasul 6: se determină și se înlătură din șir multiplii lui 2:

2 3 5 7 9 11 13 15 17 19 21 23 25 ;

- pasul 7: $k = 2$;

- pasul 8: se sare la pasul 4;

- pasul 4: $p_k = 3$;

- pasul 5: $p_k \leq \sqrt{n}$ ($3 \leq \sqrt{26}$) ?

- DA => se trece la pasul 6;

- pasul 6: se determină și se înlătură din șirul rămas multiplii lui 3:

2 3 5 7 11 13 17 19 23 25 ;

- pasul 7: $k = 3$;

- pasul 8: se sare la pasul 4;

- pasul 4: $p_k = 5$ (5 este al treilea număr rămas în șir);

- pasul 5: $p_k \leq \sqrt{n}$ ($5 \leq \sqrt{26}$) ?

- DA => se trece la pasul 6;

- pasul 6: se determină și se elimină din șirul rămas multiplii lui 5:

2 3 5 7 11 13 17 19 23 ;

- pasul 7: $k = 4$;

- pasul 8: se sare la pasul 4;

- pasul 4: $p_k = 7$;

- pasul 5: $p_k \leq \sqrt{n}$ ($7 \leq \sqrt{26}$) ?

- NU => se sare la pasul 9;

- pasul 9: numerele prime aparținând intervalului $[2, 26]$ sînt:

2 3 5 7 11 13 17 19 23

- pasul 10: stop.

Soluția este corectă.

● Conexiunile algoritmilor cu mulțimile valorilor variabilelor de intrare și de ieșire

Dacă analizăm cele două exemple prezentate, privite de data aceasta din exterior (considerînd fiecare algoritm ca o "cutie", ca un aparat) observăm că obținerea uneia sau altei soluții depinde de datele de intrare și nu de ce se întîmplă în interiorul algoritmului respectiv. Aceasta ne îndeamnă să vedem algoritmul și ca un sistem închis (figura 2.1) care are două legături cu exteriorul: una de intrare (i) și una de ieșire (o).

Pe de altă parte, așa cum am văzut, algoritmi sînt generali și reprezintă o succesiune mai simplă sau mai complicată de relații care, de fapt, nu sînt altceva decît legături între mulțimea valorilor variabilelor de intrare și mulțimea soluțiilor. Notînd cu I mulțimea valorilor variabilelor de intrare și cu O mulțimea soluțiilor putem defini algoritmul ca o funcție

$$f : I \rightarrow O .$$

Sigur, această funcție poate fi banală (dacă ne-am propune, de exemplu, să întocmim un

algoritm care să ne furnizeze distanța parcursă de un mobil în mișcare rectilinie uniformă am găsi că f nu este altceva decât:

$$f(t) = s = v \cdot t$$

adică o funcție de gradul întâi) sau ca în majoritatea cazurilor mai mult sau mai puțin complicată (vezi exemplele prezentate anterior).

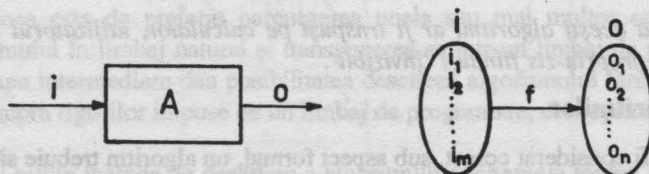


Fig. 2.1

Fără a mai complica lucrurile și fără a intra în amănunte trebuie subliniat totuși și acest mod de interpretare a algoritmului, tocmai pentru a scoate în evidență rolul, locul și interdependența dintre algoritmul propriu-zis și cele două mulțimi de valori ale variabilelor cu care operează (intrările și ieșirile).

Interacțiunea între algoritmul propriu-zis și mulțimea valorilor variabilelor de intrare are loc, de regulă, în primul (primii) pas (pași) ai săi iar legătura cu mulțimea valorilor soluțiilor se efectuează în unul (sau mai mulți) din ultimii săi pași.

Observație. La conectarea algoritmului cu mulțimea valorilor variabilelor de intrare s-a utilizat noțiunea de 'interacțiune', deoarece într-un algoritm, înainte de a începe parcurgerea operațiilor propriu-zise, este necesară verificarea condițiilor ce trebuie îndeplinite de variabilele de intrare (dacă valorile lor aparțin mulțimii valorilor admise); în cazul în care valorile acestora nu corespund condițiilor impuse, algoritmul trebuie să le 'refuze', iar parcurgerea în continuare a acestuia să fie abandonată.

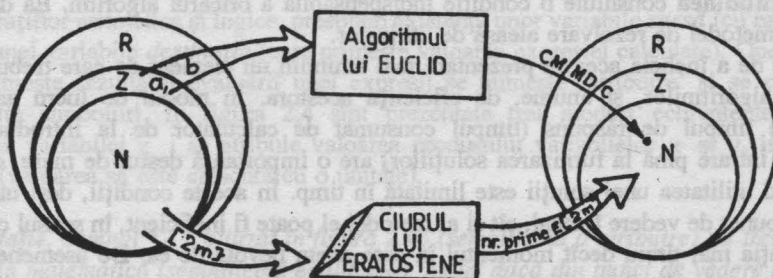


Fig. 2.2

În figura 2.2 sînt prezentate sugestiv locul și relațiile dintre cei doi algoritmi prezentați (algoritmul lui Euclid și ciurul lui Eratostene) și mulțimile valorilor variabilelor cu care acestea operează. Dacă reanalizăm algoritmul lui Euclid observăm că interacțiunea dintre acesta și mulțimea valorilor variabilelor de intrare (a și b) are loc în pașii 1-3 iar soluția este furnizată la pasul 9 (o constituind variabila de ieșire a cărei valoare aparține mulțimii soluțiilor problemei, adică $o=f(a, b)$). În mod analog stau lucrurile și în cazul variantei de algoritm prezentată pentru ciurul lui Eratostene.

Remarcă. Dacă acești algoritmi ar fi transpuși pe calculator, utilizatorul nu ar sesiza decît acești pași, algoritmul propriu-zis fiindu-i 'invizibil'.

● Proprietățile algoritmilor

Pentru a putea fi considerat corect, sub aspect formal, un algoritm trebuie să îndeplinească cel puțin trei condiții esențiale: **generalitatea, finitudinea și unicitatea.**

Generalitatea este o proprietate asupra căreia am insistat și pînă acum (mai ales în capitolul 1.3). Această proprietate, de a fi valabil pentru un gen (o clasă) de probleme, justifică însăși sensul (scopul) pentru care a fost întocmit.

Finitudinea este proprietatea unui algoritm de a permite obținerea soluției (soluțiilor) problemei după parcurgerea unui număr finit de operații. Dacă dintr-un motiv sau altul, (de obicei dintr-o eroare în logica întocmirii) în parcurgerea unui algoritm o operație (sau un grup de operații) se execută la infinit spunem că s-a intrat într-o "ciclare". Dintr-o asemenea stare algoritmul nu mai poate fi întrerupt decît printr-o intervenție exterioară.

Unicitatea este acea proprietate a unui algoritm, conform căreia ori de cîte ori se pornește de la același set de valori al variabilelor de intrare se obțin aceleași soluții. De asemenea această proprietate presupune existența unui mod cunoscut de rezolvare a fiecărei operații (acțiuni) cuprinse în algoritm. Așa cum am mai arătat și în capitolul 1.3, pentru a conferi unui algoritm această proprietate este necesară abordarea sub toate aspectele a problemei căreia îi întocmim algoritmul, astfel încît să existe variante de urmat pentru toate situațiile posibile, să fie înlăturate toate ambiguitățile.

Aceste trei proprietăți caracterizează corectitudinea unui algoritm numai sub aspect formal. Dacă un algoritm are aceste trei proprietăți nu înseamnă însă că el furnizează și soluții corecte, că este corect și din punct de vedere al problemei pentru care este întocmit.

Corectitudinea constituie o condiție indispensabilă a oricărui algoritm. Ea depinde însă de corectitudinea metodei de rezolvare aleasă de utilizator.

Înainte de a încheia această prezentare mai amintim un element de care trebuie ținut seama în întocmirea algoritmilor, și anume, de **eficiența** acestora. În modul de lucru actual (în speță conversațional) timpul de răspuns (timpul consumat de calculator de la introducerea valorilor variabilelor de intrare pînă la furnizarea soluțiilor) are o importanță destul de mare, chiar esențială, în cazurile cînd utilitatea unei soluții este limitată în timp. În aceste condiții, deși un algoritm este corect atît din punct de vedere formal, cît și al metodei el poate fi ineficient, în sensul că, de exemplu, furnizează soluția mai tîrziu decît momentul în care avem nevoie de ea. De asemenea, un algoritm poate fi ineficient și sub alte aspecte (precizie, volum de date necesare, resurse utilizate, etc.). În consecință eficiența este și ea o condiție importantă ce trebuie luată în considerare la întocmirea unui algoritm.

2.2 Descrierea algoritmilor

De obicei, cum este și cazul lucrării de față, scopul întocmirii unui algoritm este transpunerea sa pe calculator. Această transpunere se poate face direct, folosind un limbaj de programare sau parcurgând una sau mai multe etape intermediare. Exprimarea directă într-un limbaj de programare pe lângă faptul că necesită o oarecare experiență, este și destul de dificilă, mai ales în cazul algoritmilor complecși. De aceea este de preferat parcurgerea uneia sau mai multor etape intermediare între întocmirea algoritmului în limbaj natural și transpunerea sa într-un limbaj de programare.

Aceste etape intermediare dau posibilitatea descrierii algoritmului fără a ne concentra în mod deosebit atenția asupra rigorilor impuse de un limbaj de programare, ci asupra corectitudinii întocmirii acestuia.

Există mai multe metode de descriere a algoritmilor: schemele logice, pseudocodul, tabelele de decizii, diagramele, limbajele specializate de descriere, etc. Din acestea ne oprim numai asupra primelor două.

● Schemele logice

Schemele logice sînt *forme de reprezentare grafică a algoritmilor folosind simboluri a căror formă indică tipul acțiunii*. Aceste simboluri poartă numele de *blocuri*. În interiorul blocurilor, de la caz la caz, pot figura una sau mai multe operații care descriu efectiv acțiunea ce se execută.

Deși există o multitudine de simboluri utilizate în descrierea algoritmilor, blocurile fundamentale necesare întocmirii unei scheme logice sînt prezentate în figura 2.3.

Blocul terminal pune în evidență începutul și sfîrșitul unei scheme logice; spre deosebire de celelalte, blocul terminal START, respectiv STOP, pot să apară numai o singură dată într-o schemă logică. Deci, o schemă logică nu poate avea decît un singur început, respectiv un singur sfîrșit.

Blocul de intrare / ieșire pune în evidență informațiile de intrare, respectiv ieșire, precum și momentul și locul în care acestea se execută; prin intermediul acestor blocuri are loc schimbul de informații între algoritmul propriu-zis și exterior.

Blocul de calcul pune în evidență operațiile aritmetice, precum și transferul de valori între diferitele variabile utilizate în algoritm. Atît transferul de valori între diferitele variabile, cît și efectuarea operațiilor aritmetice și logice, presupun existența unor variabile *sursă* (cu care se efectuează operația) și a unei variabile *destinație* (care primește valoarea expresiei calculate). Operația prin care o variabilă primește rezultatul evaluării unei expresii se numește 'atribuire' și se poate evidenția utilizînd diferite simboluri. În figura 2.4 sînt prezentate trei moduri echivalente prin care se simbolizează că variabilei z i se atribuie valoarea produsului variabilelor x și y , iar variabilei k , valoarea $k+1$ (valoarea sa este crescută cu o unitate).

Observație. Semnul '=' utilizat în figura 2.4c (semnificînd o atribuire) nu trebuie confundat cu cel utilizat în matematică (semnificînd egalitatea). Astfel dacă din punct de vedere al operației de atribuire $k=k+1$ este o propoziție adevărată, în matematică, ea este falsă.

2. Algoritmi. Metode de reprezentare

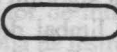




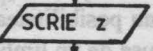
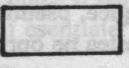
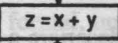
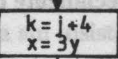

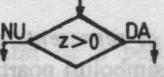
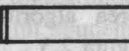
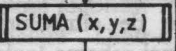



SIMBOL	SEMNICATIE	EXEMPLE	OBSERVAȚII
	bloc terminal	 	semnifică începutul schemei logice (are, numai ieșire). semnifică terminarea schemei logice (are numai intrare)
	bloc de intrare / ieșire	 	o intrare și o ieșire
	bloc de calcul	 	o intrare și o ieșire
	bloc de decizie		o intrare și mai multe ieșiri
	bloc de procedură		o intrare și o ieșire
	bloc conector		mai multe intrări și o singură ieșire
	săgeata	Evidențiază sensul de parcurgere (succesiune) a elementelor schemei logice.	

Fig. 2.3

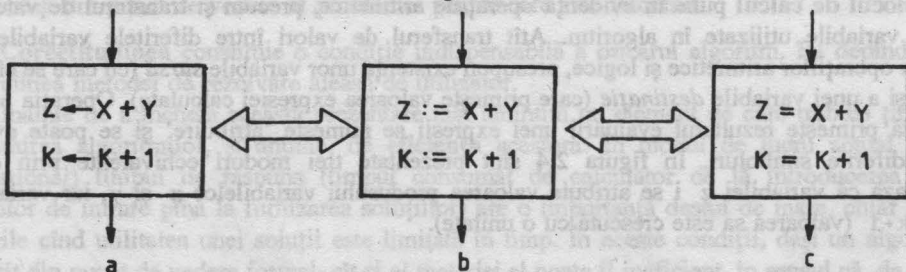


Fig. 2.4

Deoarece în limbajul BASIC, în operațiile de atribuire se utilizează semnul '=', și pentru a-l deosebi totuși de semnul '=' utilizat în matematică, în cele ce urmează vom folosi la simbolizarea

atribuirii semnul compus ':='.

Blocul de decizie pune în evidență punctele de ramificare ale algoritmului funcție de anumite condiții.

Blocul de procedură (funcție) permite scrierea prescurtată a unei secvențe care execută o anumită funcțiune; în interiorul blocului de procedură se scriu, pe lângă numele simbolic al acesteia, variabilele de intrare și de ieșire fără a face însă vreo referire la operațiile care au loc în interiorul său.

Blocul conector pune în evidență nodurile (punctele de intersecție) unui algoritm; spre deosebire de blocurile de decizie care pun în evidență căile alternative de urmat la un moment dat funcție de anumite condiții, nodurile sînt puncte în care se ajunge parcurgînd una sau alta din căile deschise de un bloc de decizie; deoarece, așa cum știm, un algoritm are un singur punct de început și unul de sfîrșit (marcate prin blocurile terminale respective) existența în cadrul acestuia a unui bloc de decizie implică automat existența a cel puțin unui bloc conector.

Modul de parcurgere al algoritmului (de trecere de la un bloc la altul) este indicat prin săgeți avînd o singură direcție (un singur vîrf). Legătura între blocuri nu poate fi deci decît univocă (condiție impusă de însăși noțiunea de algoritm).

Exemplul 2.3

O problemă foarte simplă: produsul a două numere reale. Schema logică a soluționării problemei este prezentată în figura 2.5.

Pe lângă blocurile fundamentale descrise pînă acum, în figura 2.6 sînt prezentate și alte simboluri, utilizate în special pentru a desemna suportul extern de pe care, respectiv pe care, are loc citirea, respectiv scrierea informațiilor.

● Limbajul algoritmic 'pseudocod'

Pseudocodul poate fi definit ca un *limbaj simbolic situat între limbajul natural și limbajul de programare, obținut prin introducerea unor reguli specifice limbajelor de programare în limbajul natural.*

El operează cu un set de cuvinte simbolice cu semnificații stricte utilizate în construcții sintactice care simulează structurile necesare descrierii algoritmilor. De regulă, în descrierea algoritmilor aceste cuvinte se scriu îngroșate sau subliniate, cu o aliniere distinctă față de restul textului. Aceste cuvinte mai poartă și numele de **cuvinte cheie**.

Între blocurile utilizate în schemele logice și cuvintele cheie pseudocod există o oarecare echivalență (figura 2.7).

În figura 2.8 se prezintă descrierea în pseudocod a algoritmului de calcul al produsului $z = x \cdot y$ (reprezentat și prin schema logică din figura 2.5).

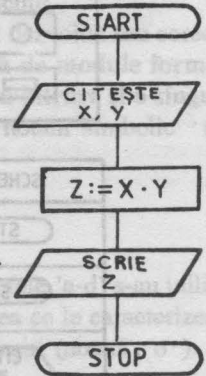


Fig. 2.5

2. Algoritmi. Metode de reprezentare

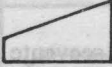
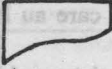
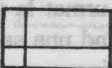




SIMBOL	SEMNIIFICAȚIE	BLOC ECHIVALENT	FUNCȚIE
	tastatură	intrare (citire)	introducerea (citirea) datelor de la tastatură
	imprimantă	ieșire (scriere)	obținerea situațiilor (scrierea) la imprimantă
	ploter	ieșire (scriere)	obținerea situațiilor grafice la ploter
	monitor (display)	ieșire (scriere)	obținerea situațiilor grafic sau alfanumeric pe ecran
	bandă magnetică (casetă magnetică)	intrare/ieșire	citirea/scrierea informațiilor într-un/dintr-un fișier pe bandă magnetică
	disc magnetic fics	intrare/ieșire	citirea/scrierea informațiilor într-un/dintr-un fișier pe disc magnetic
	floppy disk (dischetă)	intrare/ieșire	

Fig. 2.6




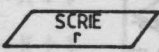
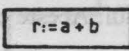
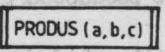
SCHEME LOGICE	PSEUDOCOD
	algoritm „nume” sau procedură „nume” etc.
	stop
	citește a, b;
	scrie r;
	r:=a+b;
	produs (a, b, c)

Fig. 2.7

algoritm produs xy;
 citește x, y;
 $z := x \cdot y$;
 scrie z;
 stop

Fig. 2.8

Observație. Caracterul ';' constituie un semn de punctuație foarte important în descrierea în pseudocod a unui algoritm. El este utilizat pentru separarea expresiilor, lipsa lui putând genera erori. Caracterul '.' semnifică alături de 'stop', sfârșitul algoritmului. Cu aceste semne de punctuație pentru descrierea unui algoritm în pseudocod este necesară și suficientă o singură frază.

Reprezentarea algoritmilor în pseudocod se caracterizează printr-o mare flexibilitate, posibilitatea exprimării algoritmilor în limbaj natural și, dacă este bine utilizat, o conversie facilă într-un limbaj de programare.

Spre deosebire de schemele logice, pseudocodul este mult mai 'bogat', el posedând expresii care, așa cum se va vedea în capitolul 2.3, simulează structurile fundamentale. El obligă utilizatorul la conceperea unor algoritmi structurați (fără salturi) ceea ce duce la o creștere substanțială a calității acestora și oferă, în același timp o bună lizibilitate și suplețe algoritmilor.

2.3 Structuri fundamentale

În cele prezentate pînă acum ne-am oprit și asupra citorva exemple de algoritmi (capitolul 2.1). Prezentarea acestora a fost efectuată în scopul familiarizării utilizatorului cu modul de abordare a unei probleme și cu noțiunea de algoritm, din care cauză descrierea acestora a fost efectuată utilizînd limbajul natural (prin enumerarea pașilor și a operațiilor de executat în fiecare moment).

În practica programării însă algoritmi se întocmesc folosind o metodă de reprezentare (capitolul 2.2) cu respectarea anumitor reguli și utilizînd un set de structuri fundamentale, astfel încît în final acesta să posede proprietățile fundamentale ale unui algoritm bine întocmit.

Pentru ca un algoritm să fie simplu, eficient, intuitiv ușor de verificat și dezvoltat sau corectat (dacă este cazul) încă de la început el trebuie realizat ca o succesiune liniară de module formate din una sau mai multe instrucțiuni (acțiuni), cu proprietatea că au o singură intrare și o singură ieșire. Astfel, dacă avem de reprezentat o secvență oarecare (pe care să o notăm simbolic 'S') într-un algoritm în care se determină valorile:

$$\begin{aligned} A &= x + y \\ B &= 2 \cdot x + y^3 \\ C &= x^2 + y^2 \end{aligned}$$

putem folosi oricare din variantele prezentate în figura 2.9, în care pentru variantele 'a-d' s-au utilizat unul sau mai multe blocuri de calcul, iar în varianta 'e', un bloc procedură. Ceea ce le caracterizează însă pe toate este existența unei singure intrări (notată 'i') și a unei singure ieșiri (notată 'o').

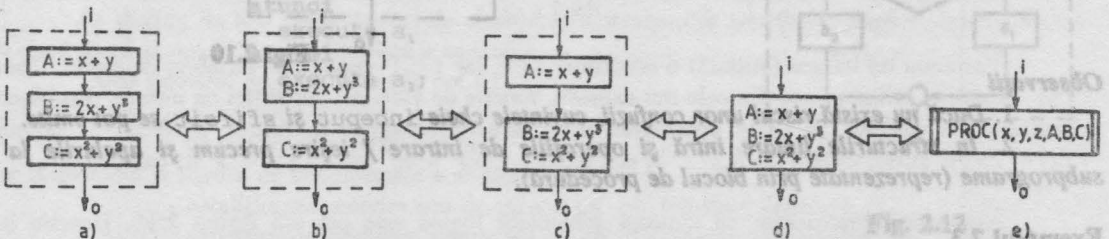


Fig. 2.9

În practică însă, în întocmirea algoritmilor întâlnim și alte condiții de derulare. Sînt astfel situații care impun, funcție de anumite condiții, execuția uneia sau altei secvențe de operații, precum și altele în care una sau mai multe operații se execută pînă se îndeplinesc anumite condiții, situații care pot determina mai multor ieșiri dintr-un modul. Cum putem ieși din această situație? Vom organiza de așa natură acțiunile în fiecare modul, astfel încît, în final să obținem o singură intrare și o singură ieșire.

Pentru aceasta dispunem de trei structuri fundamentale:

- structuri liniare;
- structuri alternative;
- structuri repetitive.

Aceste structuri constituie module avînd importanța proprietate amintită (o singură intrare și o singură ieșire).

Noțiunea de modul este însă mai largă, în sensul că un modul poate conține una sau mai multe structuri, deci poate fi, la rîndul său, alcătuit din mai multe submodule. În descrierea structurilor fundamentale vom considera însă că un modul conține o singură astfel de structură.

● Structuri liniare

Structurile liniare sînt succesiuni de operații (acțiuni) care se execută necondiționat în ordinea dată. O structură liniară care conține n operații notate a_1, a_2, \dots, a_n poate fi reprezentată, utilizînd schemele logice, ca în figura 2.10.

În pseudocod această structură se poate reprezenta astfel:

```

inceput
  executa  $a_1$ ;
  executa  $a_2$ ;
  .
  .
  executa  $a_n$ 
sfirsit.
    
```

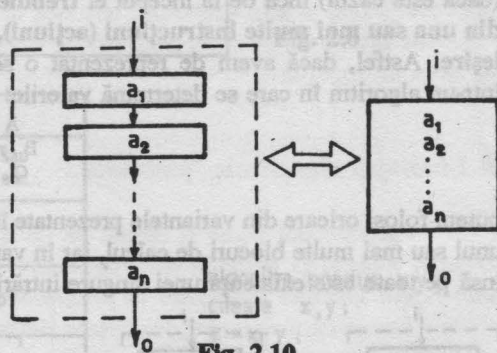


Fig. 2.10

Observații

1. Dacă nu există riscul unor confuzii, cuvintele cheie *inceput* și *sfirsit* se pot omite.
2. În structurile liniare intră și operațiile de intrare / ieșire precum și apelurile la subprograme (reprezentate prin blocul de procedură).

Exemplul 2.3

Să reprezentăm utilizînd schemele logice și în pseudocod algoritmul pentru calculul valorilor variabilelor A, B, C prezentate în figura 2.9. Algoritmul complet pentru determinarea valorilor acestor

variabile este prezentat în figura 2.11. Variabilele de intrare sînt $x, y, z \in \mathbb{R}$ (mulțimea valorilor admisibile este deci mulțimea numerelor reale), iar variabilele de ieșire sînt A, B, C .

După cum se poate observa întregul algoritm utilizează numai structuri de tip liniar.

Descris în pseudocod, algoritmul este:

```

algoritm calcul_ABC;
citeste x,y,z ;
A := x + y ;
B := 2x + y3 ;
C := x2 + y2 ;
scrie A,B,C
stop.
  
```

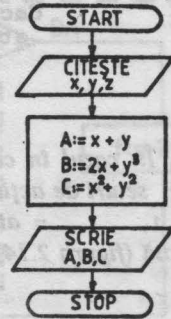


Fig. 2.11

● Structuri alternative

Cunoscute și sub numele de structuri IF-THEN-ELSE (DACA-ATUNCI-ALTFEL), și avînd forma generală prezentată în figura 2.12, sînt alcătuite dintr-un bloc de decizie (c) și două blocuri a_1 și a_2 , care se execută alternativ (sau a_1 sau a_2), după cum c este adevărată sau falsă (DA / NU), și un bloc conector.

Într-o structură alternativă derularea acțiunilor decurge astfel:

- se intră în modul i;
- se evaluează expresia c și:
 - dacă este adevărată se execută a_1 ;
 - dacă nu, se execută a_2 ;
- se iese din modul o.

După cum se poate observa, deși în interiorul modulului există două ramuri disjuncte el are totuși o singură intrare și o singură ieșire. Lucrul acesta este posibil deoarece structura alternativă începe printr-un bloc de decizie (care are o singură intrare) și se încheie printr-un bloc conector care asigură conexiunea ramurilor alternative.

În pseudocod o structură alternativă se reprezintă sub forma

```

daca c
atunci
  executa a1
altfel
  executa a2;
  
```

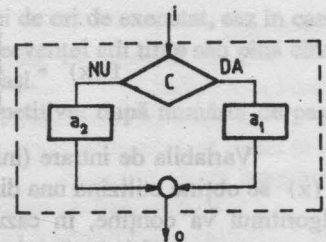


Fig. 2.12

Observații

1. Într-o structură alternativă nu este obligatorie existența ambelor blocuri a_1 și a_2 . Dacă, de exemplu a_2 nu conține operații (este vidă) atunci se obține structură alternativă cu o ramură vidă

(figura 2.13), care în pseudocod este:

daca c
atunci
executa a₁;

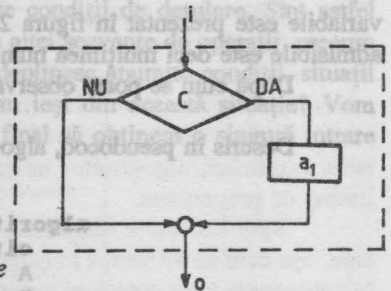


Fig. 2.13

2. În cazul în care selecția se execută dintr-o mulțime finită de n seturi de acțiuni, după un comutator k , ce ia valori în mulțimea $1, \dots, n$ atunci se obține o structură alternativă generalizată (figura 2.14), care în pseudocod se poate reprezenta astfel:

selectează k din
1: executa a_1 ;
2: executa a_2 ;
:
:
 n : executa a_n
sfîrsit.

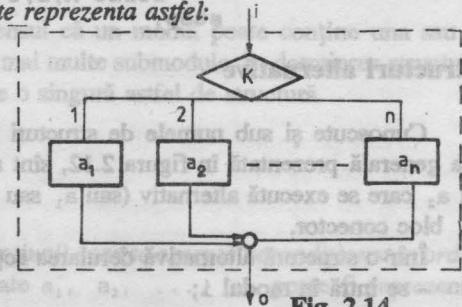


Fig. 2.14

La acest tip de structură alternativă cuvîntul cheie sfîrsit este obligatoriu.

3. Blocurile a_1 respectiv a_2 pot conține la rîndul lor structuri alternative, deci o structură alternativă poate conține la rîndul său structuri alternative.

Exemplul 2.4

Să se reprezinte cu ajutorul schemelor logice și în pseudocod algoritmul de calcul al valorilor funcției:

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f(x) = \begin{cases} -2x - 1 & \text{dacă } x < -1 \\ x^2 & \text{dacă } -1 \leq x < 1 \\ x & \text{dacă } x \geq 1 \end{cases}$$

Variabila de intrare (inițială) o constituie $x \in \mathbb{R}$, iar $f(x)$ constituie variabila de ieșire. Cum $f(x)$ se obține utilizînd una din cele trei expresii, funcție de valoarea variabilei de intrare, rezultă că algoritmul va conține, în cazul considerat, două structuri alternative. Astfel, dacă $x < -1$ atunci $f(x) = -2x - 1$, altfel, testăm dacă $x < 1$, caz în care $f(x) = x^2$ sau $f(x) = x$ în caz contrar.

Algoritmul reprezentat cu ajutorul schemelor logice este cel din figura 2.15. Descriș în pseudocod algoritmul este:

algoritm funcție
citeste x ;

```

daca (x < -1)
atunci
  f := -2x-1
altfel
  daca (x<1)
  atunci
    f := x2
  altfel
    f := x
  sfirsit ;
sfirsit ;
scrie f ;
stop.

```

Observăm că a doua structură alternativă este inclusă în prima, dar că, fiecare, privită din exteriorul său, își păstrează proprietatea fundamentală de a avea o singură intrare și o singură ieșire.

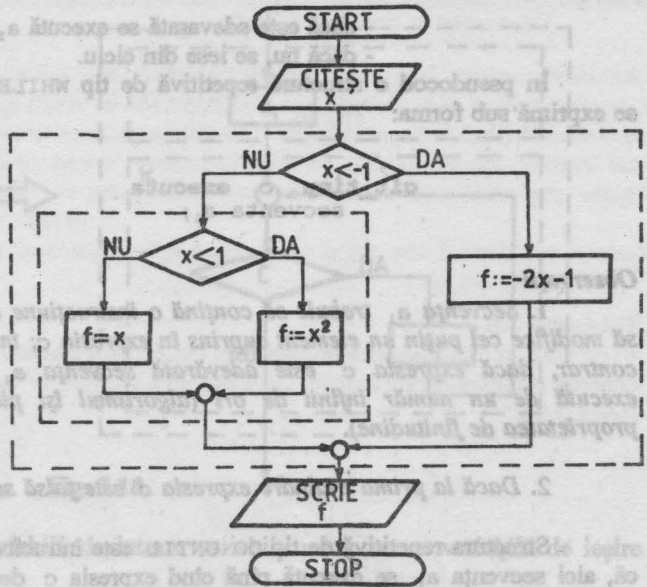


Fig. 2.15

Remarcă. Utilizarea de două ori a cuvântului cheie *sfirsit* pentru a desemna încheierea celor două structuri alternative nu este absolut necesară; au fost însă folosite pentru a mări lizibilitatea algoritmului.

● Structuri repetitive

Structurile repetitive permit efectuarea uneia sau a mai multor operații (acțiuni) de un număr de ori sau pînă cînd se îndeplinește o anumită condiție. Structurile repetitive mai poartă și denumirea de structuri ciclice sau cicluri.

Aceste structuri presupun fie cunoașterea și precizarea numărului de ori de executat, caz în care avem de-a face cu un **ciclu cu număr cunoscut de pași**, fie execuția secvenței atît timp sau pînă cînd o anumită condiție este îndeplinită, ciclul cu număr necunoscut de pași.

Se disting de aici două moduri de clasificare a structurilor repetitive: după numărul de pași și după locul și modul de condiționare a execuției.

După numărul de pași, ciclurile pot fi:

- cu număr cunoscut de pași;
- cu număr necunoscut de pași,

iar după locul și modul de condiționare a execuției sînt:

- condiționate anterior sau de tip WHILE-DO (cît-timp...execută...);
- condiționate posterior sau de tip DO-UNTIL (execută...pînă-cînd...).

Structura de tip WHILE-DO este prezentată în figura 2.16 și se execută astfel:

- se evaluează expresia c și :

- dacă este adevărată se execută a_1 și se reia;
- dacă nu, se iese din ciclu.

În pseudocod o structură repetitivă de tip WHILE-DO se exprimă sub forma:

```
cit_timp c executa
    secvența a1;
```

Observații

1. Secvența a_1 trebuie să conțină o instrucțiune care să modifice cel puțin un element cuprins în expresia c ; în caz contrar, dacă expresia c este adevărată secvența a_1 se execută de un număr infinit de ori (algoritmul își pierde proprietatea de finitudine).

2. Dacă la prima evaluare expresia c este falsă secvența a_1 nu se execută niciodată.

Structura repetitivă de tip DO-UNTIL este întrucâtva inversă structurii WHILE-DO, în sensul că, aici secvența a_1 se execută pînă cînd expresia c devine adevărată (deci atît timp cît c este falsă), locul în care se face această evaluare fiind de data aceasta la sfîrșit. Structura repetitivă DO-UNTIL este prezentată în figura 2.17 și se execută astfel:

- se execută secvența a_1 ;
- se evaluează expresia c și:
 - dacă este falsă se reia secvența a_1 ;
 - dacă este adevărată se iese din ciclu.

În pseudocod această structură se exprimă:

```
executa
    secvența a1
pina_cind c.
```

Observații

1. Și în acest caz secvența a_1 trebuie să determine modificarea condiției c ; în caz contrar ciclul este infinit.
2. Spre deosebire de structura WHILE-DO aici secvența a_1 se execută cel puțin o dată (chiar și în cazul în care expresia c este adevărată); din această cauză se recomandă utilizarea cu atenție a acestui tip de structură.
3. O structură de tip DO-UNTIL poate fi simulată utilizînd o structură liniară (1) și una de tip WHILE-DO (w), figura 2.18.

Exemplul 2.5

Să se descrie utilizînd schemele logice și în pseudocod, un algoritim pentru determinarea sumei primelor n numere naturale.

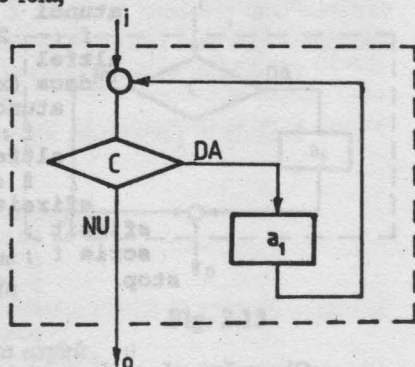


Fig. 2.16

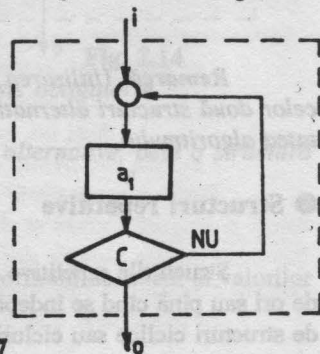


Fig.2.17

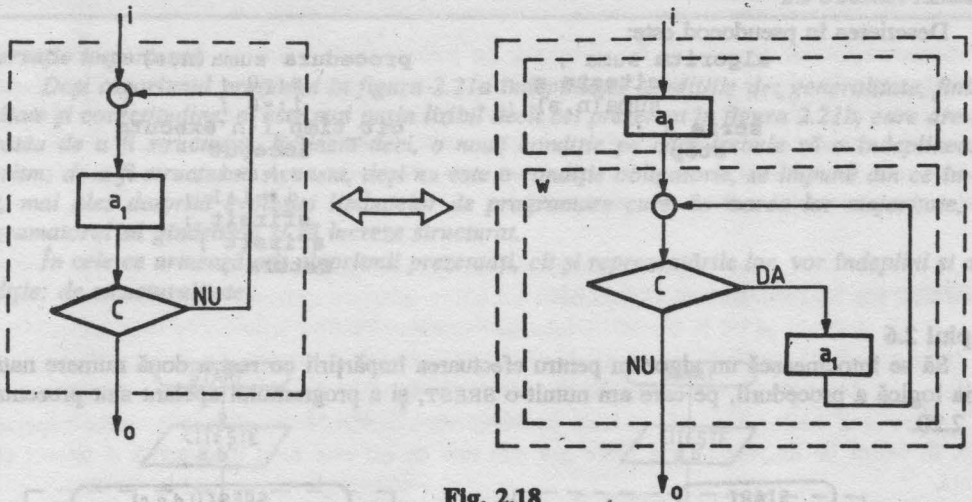


Fig. 2.18

Pentru aceasta vom utiliza ca variabilă de intrare variabila $n \in \mathbb{N}$, iar ca variabilă de ieșire variabila $s = \sum_{i=1}^{n-1} i$ (deoarece primul număr natural este 0, atunci primele n numere naturale vor fi cuprinse în intervalul $[0, n-1]$).

Numărul n fiind cunoscut se va utiliza pentru determinarea sumei o structură cu număr cunoscut de pași. Suma s se determină în procedura SUMA.

Schema logică a algoritmului este prezentată în figura 2.19 blocurile terminale SUMA(n, s) și respectiv RETURN reprezintă intrarea respectiv revenirea din procedură.

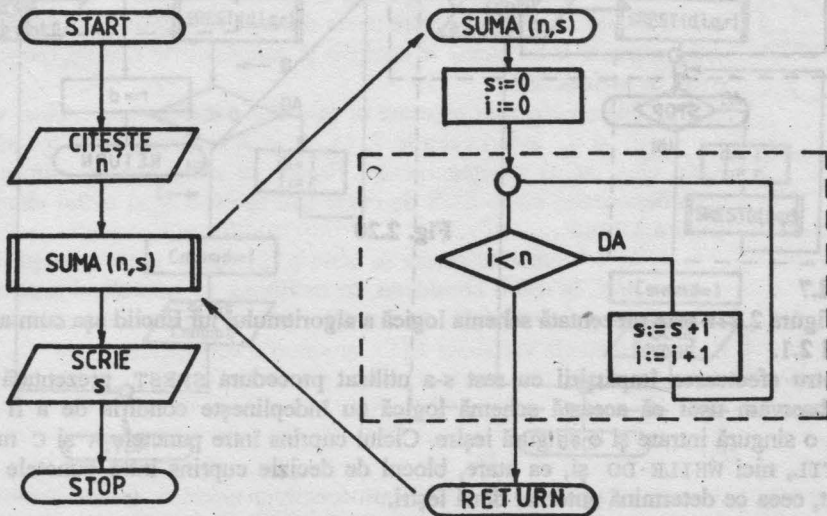


Fig. 2.19

Descrierea în pseudocod este:

```

algoritm suma ;
    citește n ;
    suma(n,s) ;
    scrie s ;
    stop.
    
```

```

procedura suma(n,s)
    s:=0 ;
    i:=0 ;
    cit_timp i<n executa
    început
    s:=s+i ;
    i:=i+1 ;
    sfîrșit ;
    sfîrșit ;
    return ;
    
```

Exemplul 2.6

Să se întocmească un algoritm pentru efectuarea împărțirii cu rest a două numere naturale. Schema logică a procedurii, pe care am numit-o SREST, și a programului apelant sînt prezentate în figura 2.20.

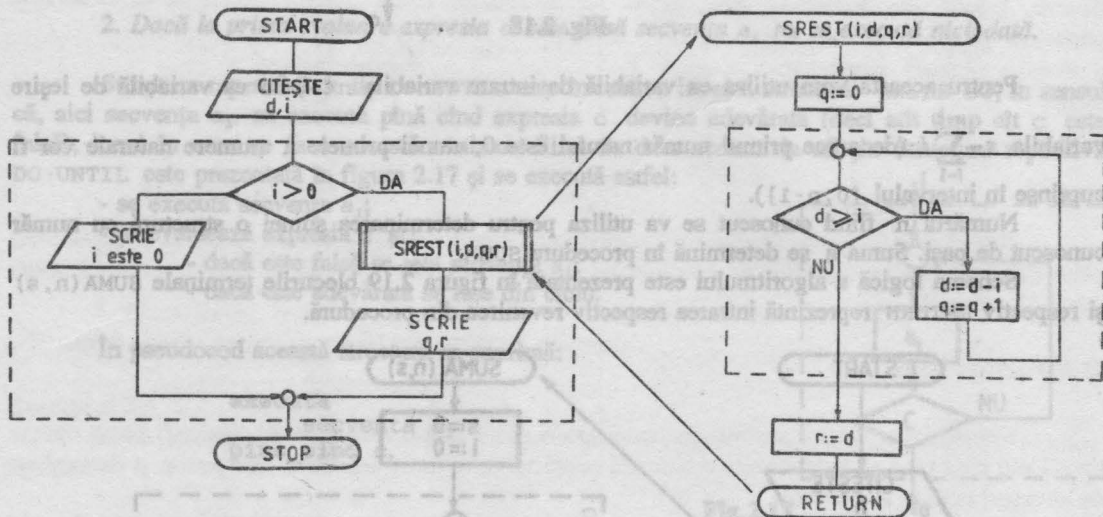


Fig. 2.20

Exemplul 2.7

În figura 2.21a este prezentată schema logică a algoritmului lui Euclid așa cum a fost întocmit în exemplul 2.1.

Pentru efectuarea împărțirii cu rest s-a utilizat procedura SREST prezentată în exemplul anterior. Observăm ușor că această schemă logică nu îndeplinește condiția de a fi alcătuită din structuri cu o singură intrare și o singură ieșire. Ciclul cuprins între punctele A și C nu este nici de tip DO-UNTIL, nici WHILE-DO și, ca atare, blocul de decizie cuprins între punctele B și C este independent, ceea ce determină automat două ieșiri.

Pentru a obține o schemă structurată este necesară o mică modificare, astfel încât se obține schema logică din figura 2.21b.

Observație importantă

Deși algoritmul prezentat în figura 2.21a îndeplinește condițiile de: generalitate, finitudine, unicitate și corectitudine, el este mai puțin lizibil decât cel prezentat în figura 2.21b, care are în plus calitatea de a fi structurat. Aceasta, deși nu este o condiție obligatorie, se impune din ce în ce mai mult, mai ales datorită evoluției limbajelor de programare care, în marea lor majoritate, obligă programatorul să gândească și să lucreze structurat.

În cele ce urmează ați algoritmiu prezentați, cit și reprezentările lor, vor îndeplini și această condiție: de structuralitate.

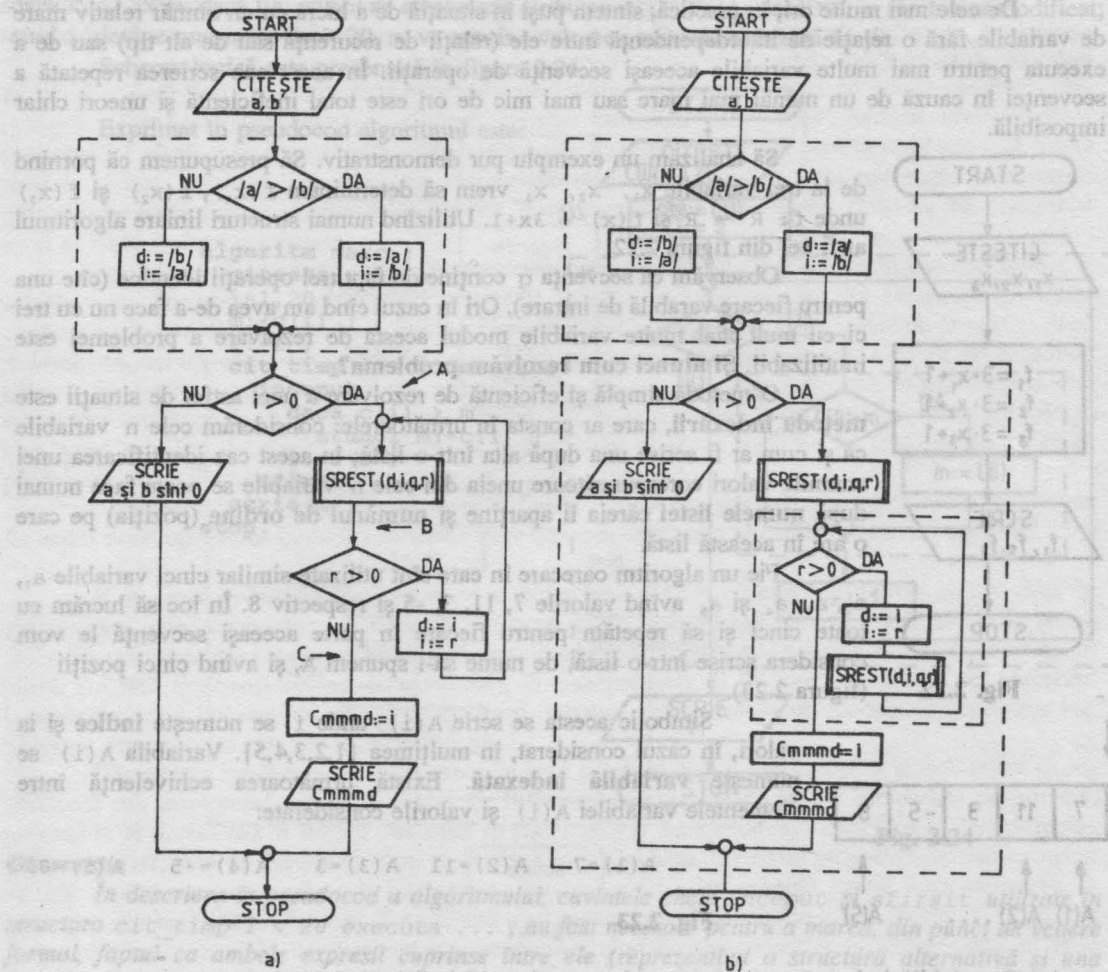


Fig. 2.21

Remarcă finală. Structurile fundamentale nu trebuie confundate cu blocurile fundamentale utilizate în reprezentarea algoritmilor cu ajutorul schemelor logice. Dacă blocurile de calcul, blocurile de intrare / ieșire și blocurile procedură pot fi considerate și ca structuri de tip liniar, blocul de decizie, avînd o intrare și cel puțin două ieșiri nu poate fi asimilat cu nici una din structuri. În mod asemănător stau lucrurile și în cazul blocului conector. Blocurile de decizie și conector intră însă în compunerea structurilor alternative și repetitive.

2.4 Utilizarea variabilelor indexate

De cele mai multe ori, în practică, sîntem puși în situația de a lucra cu un număr relativ mare de variabile fără o relație de interdependență între ele (relații de recurență sau de alt tip) sau de a executa pentru mai multe variabile aceeași secvență de operații. În acest caz scrierea repetată a secvenței în cauză de un număr mai mare sau mai mic de ori este total ineficientă și uneori chiar imposibilă.

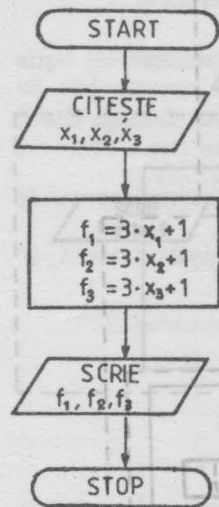


Fig. 2.22

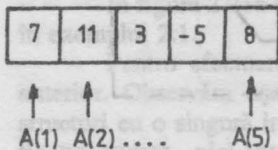
Să analizăm un exemplu pur demonstrativ. Să presupunem că pornind de la trei variabile x_1, x_2, x_3 vrem să determinăm $f(x_1), f(x_2)$ și $f(x_3)$ unde $f: \mathbb{R} \rightarrow \mathbb{R}$ și $f(x) = 3x + 1$. Utilizînd numai structuri liniare algoritmul ar fi cel din figura 2.22.

Observăm că secvența q conține de fapt trei operații identice (cite una pentru fiecare variabilă de intrare). Ori în cazul cînd am avea de-a face nu cu trei ci cu mult mai multe variabile modul acesta de rezolvare a problemei este inutilizabil. Și atunci cum rezolvăm problema?

O metodă simplă și eficientă de rezolvare a unei astfel de situații este metoda indexării, care ar consta în următoarele: considerăm cele n variabile ca și cum ar fi scrise una după alta într-o listă; în acest caz identificarea unei anumite valori corespunzătoare uneia din cele n variabile se poate face numai după numele listei căreia îi aparține și numărul de ordine (poziția) pe care o are în această listă.

Fie un algoritm oarecare în care sînt utilizate similar cinci variabile a_1, a_2, a_3, a_4 și a_5 avînd valorile 7, 11, 3, -5 și respectiv 8. În loc să lucrăm cu toate cinci și să repetăm pentru fiecare în parte aceeași secvență le vom considera scrise într-o listă, de nume să-i spunem A , și avînd cinci poziții (figura 2.23).

Simbolic acesta se scrie $A(i)$ unde i se numește indice și ia valori, în cazul considerat, în mulțimea $\{1, 2, 3, 4, 5\}$. Variabila $A(i)$ se numește variabilă indexată. Există următoarea echivalență între elementele variabilei $A(i)$ și valorile considerate:



$$A(1) = 7 \quad A(2) = 11 \quad A(3) = 3 \quad A(4) = -5 \quad A(5) = 8.$$

Fig. 2.23

O variabilă indexată poate avea unul sau mai mulți indici. Variabilele indexate cu un singur indice se mai numesc tablouri unidimensionale sau vectori, iar cele cu doi indici, de exemplu $B(i, j)$, tablouri bidimensionale sau matrici.

Exemplul 2.8

Să se întocmească un algoritm pentru determinarea elementului maxim dintr-un șir de 20 de valori date.

Pentru aceasta utilizăm variabila indexată $C(i)$ unde:

$$i \in \{i / 1 \leq i \leq 20 \text{ și } i \in \mathbb{N}\}.$$

Variabila $C(i)$ constituie variabila de intrare. Variabila de ieșire este $m = \max C(i)$. Pentru determinarea elementului maxim, atribuim inițial lui m valoarea primului element din listă (vectorul $C(m=C(1))$), apoi modificând pe rând valoarea indicelui i între 2 și 20 se verifică, de fiecare dată, dacă $C(i) > m$; dacă DA atunci se efectuează atribuire $m := C(i)$, dacă NU, m rămâne nemodificat; când i devine mai mare decât 20, m va conține valoarea maximă existentă în șir.

Schema logică este prezentată în figura 2.24.

Exprimat în pseudocod algoritmul este:

```

algoritm maxc ;
citeste c(i) ;
i=1,20
m:=c(i) ;
i:=2 ;
cit timp i < 20 executa
inceput
daca c(i) > m
atunci m:=c(i) ;
i:=i+1 ;
sfirsit ;
scrie m ;
stop.
  
```

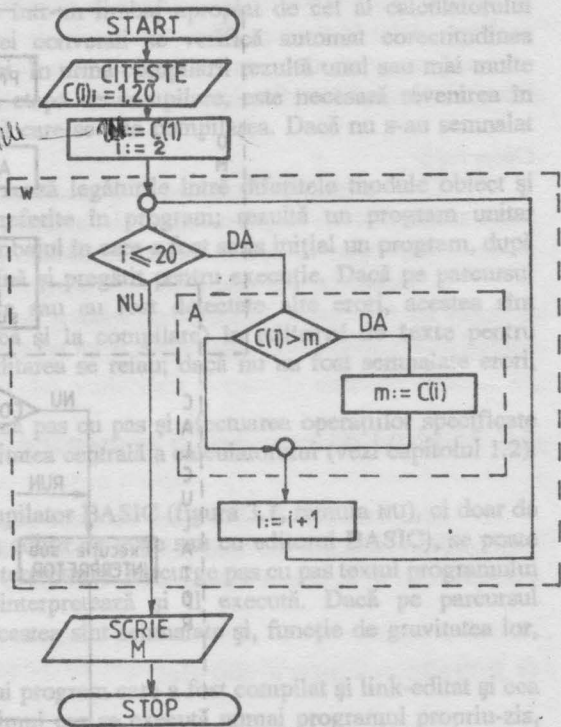


Fig. 2.24

Observație

În descrierea în pseudocod a algoritmului, cuvintele cheie **inceput** și **sfirsit** utilizate în structura **cit timp** $i < 20$ **executa** ..., au fost necesare pentru a marca, din punct de vedere formal, faptul ca ambele expresii cuprinse între ele (reprezentind o structură alternativă și una liniară) se repetă atît timp cît condiția $i < 20$ este îndeplinită. Dacă acestea ar fi lipsit, ar fi rezultat că se repetă numai prima expresie (respectiv structura alternativă), ceea ce este eronat.

3. ELEMENTE ALE LIMBAJULUI BASIC

3.1 Cîteva precizări

În general, scopul final al elaborării unui algoritm este transpunerea și execuția automată a acestuia cu ajutorul calculatorului. În esență, etapele de parcurs pentru îndeplinirea acestui deziderat sînt prezentate în figura 3.1.

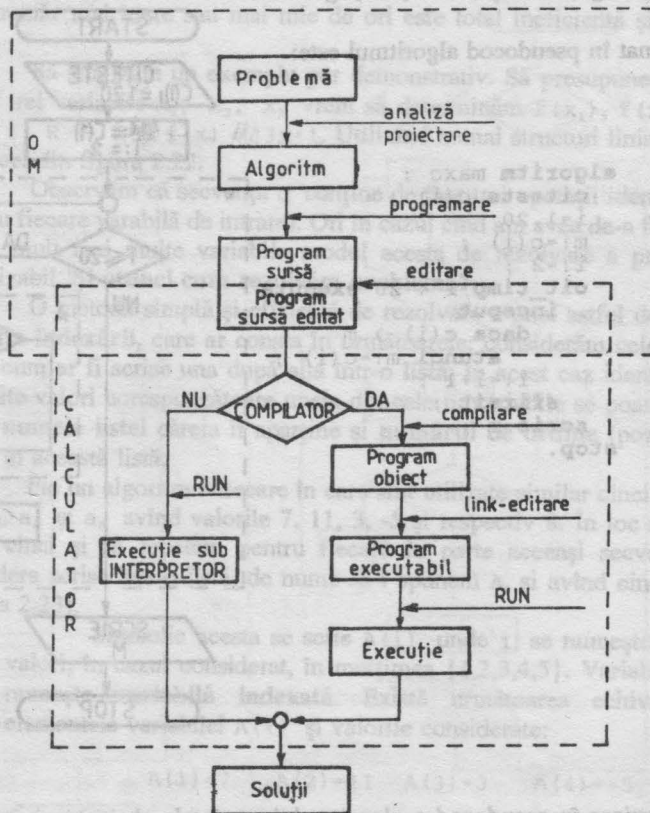


Fig. 3.1

Am văzut, în capitolele anterioare, care sînt principalele cerințe și cum putem întocmi algoritmul soluționării unei probleme.

După cum rezultă și din figura 3.1, pentru a transpune algoritmul pe calculator, pentru a putea

fi 'înțeles' și executat de acesta, ar mai fi de parcurs cel puțin încă o etapă, și anume, descrierea sa într-un limbaj de programare. Am ales limbajul BASIC.

Înainte de a trece însă la descrierea efectivă a elementelor de bază ale limbajului BASIC sînt necesare cîteva precizări referitoare la fazele prin care trece un program de la introducerea pînă la execuția sa efectivă.

O primă fază constă în editarea (introducerea de la tastatură - scrierea) programului în format sursă (așa cum a fost scris) și reținerea lui pe un suport extern de informații (bandă sau disc magnetic). Lucrul acesta se execută cu ajutorul unui editor de texte, program sistem de regulă, care permite introducerea, încărcarea și salvarea (reținerea) unui text (respectiv textul programului).

După editarea și eventual corectarea textului programului, funcție de modul de lucru (de varianta de BASIC de care dispunem, compilator sau interpretor) se poate proceda în două moduri:

A. Cînd se dispune de compilator se parcurg fazele (figura 3.1, ramura DA):

- **compilarea** programului, care constă în conversia unui program din limbajul de programare de nivel înalt în care a fost scris (program sursă) într-un limbaj apropiat de cel al calculatorului (program obiect); înainte sau pe parcursul acestei conversii se verifică automat corectitudinea programului sursă și se semnalează eventualele erori. În urma compilării rezultă unul sau mai multe module obiect. Dacă au fost detectate erori, după etapa de compilare, este necesară revenirea în editorul de texte pentru efectuarea corecțiilor, după care se reia compilarea. Dacă nu s-au semnalat erori se trece la faza următoare.

- **editarea de legături** (link-editarea) realizează legăturile între diferitele module obiect și soluționează apelurile la funcțiile de bibliotecă referite în program; rezultă un program unitar reprezentat în cod direct executabil; indiferent de limbajul în care a fost scris inițial un program, după link-editare, acesta este tradus complet în cod mașină și pregătit pentru execuție. Dacă pe parcursul editării de legături au rămas referiri nesoluționate sau au fost detectate alte erori, acestea sînt semnalate automat; în acest caz trebuie revenit (ca și la compilare) în editorul de texte pentru corectarea erorilor, după care compilarea și link-editarea se reiau; dacă nu au fost semnalate erori, programul poate fi lansat în execuție;

- **execuția** programului constă în parcurgerea pas cu pas și efectuarea operațiilor specificate de codul instrucțiunii, în limbaj mașină, de către unitatea centrală a calculatorului (vezi capitolul 1.2).

B. În cazul în care nu se dispune de un compilator BASIC (figura 3.1, ramura NU), ci doar de interpretor, după introducerea textului sursă (cu un editor de texte sau cu editorul BASIC), se poate trece direct la execuția programului. În acest caz, 'interpretorul' parcurge pas cu pas textul programului sursă BASIC (instrucțiune cu instrucțiune), îl interpretează și îl execută. Dacă pe parcursul interpretării unei instrucțiuni sînt detectate erori, acestea sînt semnalate și, funcție de gravitatea lor, execuția programului este abandonată sau nu.

Deosebirea fundamentală între execuția unui program care a fost compilat și link-editat și cea efectuată sub interpretor, constă în faptul că, în primul caz se execută numai programul propriu-zis, pe cînd în al doilea caz în timpul execuției se efectuează și interpretarea, verificarea și decodificarea instrucțiunilor, ceea ce duce la un timp de răspuns mult mai lung. Cu cît un program este mai complex cu afit durata execuției sale sub interpretor este mai mare. Din această cauză interpretoarele, la execuții, sînt mai puțin eficiente. Totuși, mai ales pentru începători sau în etapa de testare și punere la punct a unor module de program sau programe, cînd durata execuției nu interesează în mod deosebit, interpretorul asigură o mai bună flexibilitate și intervale de timp mai scurte între efectuarea unei corecții în programul sursă și verificarea eficienței acesteia în execuție.

În urma execuției programului se obțin soluțiile problemei.

Continua evoluție a limbajelor de programare, mai ales după apariția și dezvoltarea de amploare a calculatoarelor personale, a dus la apariția așa numitelor 'medii de programare', mai întâi pentru limbajul PASCAL și apoi treptat și pentru alte limbaje, inclusiv BASIC.

Mediile de programare sînt sisteme integrate de programe care asigură editarea, compilarea, link-editarea, execuția și depanarea programelor. Ele oferă marele avantaj că, printr-un set de comenzi proprii, asigură toate funcțiile necesare punerii la punct a programelor, ceea ce duce la o programare deosebit de eficientă. Indiferent de varianta de BASIC și de modul de lucru (sub interpretor, cu compilator sau în mediu de programare), în esență, nucleul vocabularului (vocabularul fundamental), semantica (setul de reguli care determină înțelesul cuvintelor - al cuvintelor cheie - și semnificația propozițiilor) și regulile sintactice (de combinare a cuvintelor în propoziții, și relațiile dintre acestea la nivelul frazei - așa cum s-a arătat în capitolul 2, un program poate fi asimilat cu o frază) sînt aceleași, astfel încît, cunoscînd elementele de bază, conceptele și structurile fundamentale cu care operează, se poate trece cu ușurință de la o versiune de BASIC la alta, atît din punct de vedere al însușirii, cît și al transferului programelor.

Observații.

1. *O prezentare exhaustivă a multitudinii variantelor de BASIC existente în prezent, a particularităților de implementare ale acestora pe diferite tipuri de calculatoare, precum și a modurilor specifice de operare este efectuată în [4].*

2. *Deoarece obiectul prezentei lucrări nu este prezentarea limbajului BASIC ci utilizarea acestuia ca instrument în însușirea tehnicii de programare a calculatoarelor în general, în acest capitol se prezintă numai elementele strict necesare atingerii acestui obiectiv.*

3.2 Moduri de operare și comenzile interpretorului

Interpretoarele BASIC fiind mult mai răspîndite decît compilatoarele vom începe prezentarea principalelor elemente ale limbajului BASIC cu cîteva comenzi, minim necesare lucrului cu acesta, cu exemplificări în BASIC-80 (MBASIC, GBASIC) și GW-BASIC <G>.

● Inițializarea interpretorului

Se realizează de obicei tastînd, ca o comandă a sistemului de operare, numele acestuia.

De exemplu, apelul interpretorului MBASIC sub CP/M se efectuează cu comanda:

```
MBASIC [fis] [/F:maxf] [/M:maxm] [S:maxr]
```

unde

fis este numele fișierului avînd implicit tipul (extensia) BAS care, după încărcare, este lansat în execuție;

maxf reprezintă numărul maxim de fișiere de date simultan deschise;
maxm este limită maximă a memoriei pe care o poate folosi interpretorul;
maxr reprezintă lungimea maximă a înregistrării din fișierele de date.

Interpretorul GW-BASIC, sub MS-DOS se apelează cu:

```
GWBasic [fis] [, stdin] [, stdout] [/F:maxf] [/S:maxr]
        [/M:maxm] [, lgb] [/D] [/I]
```

unde

stdin și **stdout**

desemnează eventualele redirectări ale fișierelor standard de intrare, respectiv ieșire;

lgb specifică dimensiunea spațiului de lucru rezervat pentru date și programe încărcate peste spațiul de lucru al lui GW-BASIC;

D încarcă în memorie pachetele de rutine matematice pentru funcții transcendente;

I determină alocarea statică a spațiului de memorie necesar pentru operații la nivel de fișier.

După inițializare se tipărește prompterul **Ok** semnificînd faptul că interpretorul este operațional.

● Moduri de operare

După lansare, interpretorul poate fi utilizat în două moduri: **direct** și **indirect**.

Modul direct permite introducerea comenzilor și instrucțiunilor fără număr de linie și asigură execuția acestora imediat după tastarea caracterului <CR> (RETURN, ENTER). Deși instrucțiunile nu sînt memorate (după execuție linia introdusă se pierde), variabilele și valorile atribuite acestora sînt reținute în memoria de lucru a interpretorului.

Modul de operare direct este util pentru depanarea unor secvențe liniare, precum și pentru efectuarea unor operații rapide, care nu necesită un program complet.

Observație. În acest mod de operare pot fi utilizate numai instrucțiunile ce nu presupun salturi de orice fel către alte linii BASIC.

Modul indirect permite introducerea unui program linie cu linie. O linie program BASIC, funcție de familie, versiune, generație, poate conține, de regulă, între 80 și 255 caractere și are sintaxa:

```
etich instr1 [:instr2 ...] <CR>
```

unde

etich este o valoare numerică întregă cuprinsă între 1 și o valoare maximă (32000, 32767, 65529 etc.) funcție de versiune, și are două utilizări: arată ordinea de memorare a liniilor program și poate fi referită în instrucțiunile de salt;

instr este o propoziție care reprezintă descrierea în BASIC a unei operații;

: reprezintă semnul de punctuație utilizat în BASIC pentru separarea instrucțiunilor în cazul în

care sînt mai multe pe aceeași linie.

Sfîrșitul unei linii este semnalat de caracterul <CR>.

O linie BASIC (în majoritatea versiunilor) poate conține mai multe linii fizice, trecerea, în acest caz, la altă linie fizică efectuîndu-se cu caracterul <LF> sau combinația <CTRL>J.

Observație. În cazul în care într-o linie BASIC este specificată o singură instrucțiune, separatorul : nu mai este necesar, rolul său fiind îndeplinit de caracterul <CR>. Observația este valabilă și în cazul existenței mai multor instrucțiuni pe aceeași linie, pentru instrucțiunea ce precede caracterul <CR>.

Numerele liniilor pot fi generate automat. Pentru aceasta se utilizează comanda:

```
<B,G>
AUTO      [nrl] [,incr]
```

Generarea începe cu numărul nrl, continuînd cu pasul (incrementul) incr (implicit nrl=incr=10); revenirea în mod comandă se efectuează cu <CTRL>C, respectiv <CTRL><BREAK> în GW-BASIC.

Programul introdus rămîne în memorie, asupra lui putîndu-se efectua o serie de operații, pînă la ștergerea sau la revenirea în sistemul de operare.

● Salvări, restaurări

Salvarea programului curent într-un fișier disc fis se realizează cu comanda:

```
<B,G>
SAVE      "fis" [,A] [,P]
```

Dacă fis există, atunci programul salvat va fi scris peste acesta; opțiunea A determină salvarea programului în format ASCII, iar P duce la protecția (la listare sau editare) fișierului salvat.

Încărcarea în memorie a unui fișier disc fis se efectuează cu comanda:

```
<B,G>
LOAD      "fis" [,R]
```

Prezența opțiunii R determină, după încărcare, lansarea automată în execuție a programului.

O altă comandă care permite încărcarea în memorie a unui fișier disc este:

```
<B,G>
MERGE     "fis"
```

Spre deosebire de LOAD care determină înaintea încărcării ștergerea programului curent, MERGE combină programul curent cu fișierul disc fis. Dacă cele două programe au linii comune, liniile respective ale programului fis le înlocuiesc pe cele corespunzătoare din programul curent.

Observație importantă. Deoarece pe parcursul lucrării de față vom utiliza frecvent această comandă, subliniem faptul că alături de comenzile LOAD și SAVE, MERGE face parte și din setul de

comenzi ale următoarelor interpretoare: BASIC HC-85, TIM S, SPECTRUM, BASIC-AMSTRAD, ABASIC, MBASIC 86, ZBASIC, BASIC-TI, BETA-BASIC. În cazul microcalculatoarelor care nu dispun de unitate de dischetă, comanda MERGE va determina încărcarea în memoria internă a programului fis înregistrat pe caseta magnetică. Ca și în cazul comenzilor LOAD și SAVE, înaintea lansării comenzii MERGE pregătirea casetofonului și eventuala poziționare a benzii magnetice se vor efectua conform particularităților de utilizare a timpului de calculator de care se dispune.

Interpretorul GW-BASIC dispune de încă două comenzi pentru salvarea (BSAVE), respectiv restaurarea (BLOAD) unei zone de memorie.

Comanda

```
<G>
BSAVE "fis",depl,lg
```

salvează zona de memorie de lg octeți, care începe de la adresa relativă depl (deplasarea în octeți) față de segmentul definit de un DEF SEG anterior (implicit segmentul de date al interpretorului GW-BASIC), în fișierul disc fis.

Comanda

```
<G>
BLOAD "fis"[,depl]
```

încarcă în memorie fișierul în cod mașină (imagine memorie) fis la adresa relativă depl față de adresa segmentului definit de ultimul DEFSEG sau, în lipsa acestuia, față de adresa segmentului de date al interpretorului GW-BASIC.

● Ștergeri, modificări, listări

Ștergerea programului și a tuturor variabilelor existente în memorie se efectuează cu comanda:

```
<B,G>
NEW
```

Ștergerea unui fișier oarecare fis de pe disc se realizează cu:

```
<B,G>
KILL "fis"
```

Observație. Fișierul fis trebuie să fie închis.

Ștergerea uneia sau mai multor linii, având numerele cuprinse între nr1 și nr2, inclusiv se realizează cu:

```
<B,G>
DELETE [nr1] [-] [nr2]
```

Modificarea (în mod imediat) a unei linii având numărul nr1 se poate efectua cu comanda:

```
<B,G>
EDIT [nr1]
```

Aceasta determină intrarea în modul editare. Modificarea efectivă a textului instrucțiunii respective se realizează cu un set de subcomenzi, care în cazul interpretorului MBASIC sînt:

Spațiu	mută cursorul la dreapta;
<RUBOUT>	mută cursorul la stînga;
I<text>	inserează 'text' pe poziția curentă a cursorului. <text> se poate încheia cu <ESC>, caz în care editarea continuă sau cu <CR>, caz în care se iese din EDIT;
X	inserare la sfîrșit de linie;
[n]D	șterge n caractere la dreapta cursorului (inclusiv poziția curentă);
H	șterge pînă la sfîrșitul liniei;
[n]S<ch>	caută a n-a apariție a caracterului ch și poziționează cursorul înaintea acestuia;
[n]K<ch>	șterge a n-a apariție a caracterului ch;
[n]C<ch>	înlocuiește următoarele n caractere cu ch;
<CR>	salvează modificările făcute în linie, iese din modul EDIT și tipărește restul liniei;
E	același efect ca <CR>, însă fără tipărire;
Q	dă controlul interpretorului MBASIC după salvarea modificărilor efectuate cu EDIT;
L	listează restul liniei și poziționează cursorul la începutul acesteia;
A	reface linia inițială și poziționează cursorul la începutul ei.

Schimbarea numelui unui fișier disc se realizează cu:

```
<B,G>
NAME "numen" AS "numev"
```

Fișierul numev, dacă există, primește numele numen.

Renumerotarea liniilor se efectuează cu comanda:

```
<B,G>
RENUM [nr11] [,nr12] [,incr]
```

unde nr11 reprezintă noul număr de la care începe numerotarea liniilor, număr ce se atribuie celei ce avea eticheta nr12; incr, dacă este prezent, semnifică pasul incrementării; dacă nu, acesta este 10.

Observație. RENUM actualizează și argumentele instrucțiunilor care fac trimeri la numere de linii mai mari sau egale cu nr12.

Listarea programului existent în memorie se poate efectua cu comenzile:

```
<B>
LIST [nr11] [-] [nr12]
```

```
<G>
LIST [nr11] [-] [nr12] [flis]
```



```
<B,G>
LLIST      [nr1] [-] [nr2]
```

Acestea permit listarea unei părți (liniile de la nr1 la nr2 inclusiv) sau a întregului program existent în memorie (LIST fără nr1 și nr2) la terminal (MBASIC), în fișierul de listare (GW-BASIC, dacă flis este specificat) sau la imprimantă (LLIST).

De asemenea, în GW-BASIC, tipărirea conținutului ecranului pe imprimanta implicită se poate comanda cu:

```
<G>
LCOPY
```

● Execuții

Comanda

```
<B,G>
RUN      [nr1]
```

lansează în execuție un program aflat în memorie începînd cu linia nr1. Dacă nr1 nu este precizat atunci execuția începe cu prima linie executabilă a programului.

În forma

```
<B,G>
RUN      "fis" [,R]
```

are loc ștergerea conținutului memoriei, încărcarea fișierului fis de pe disc și lansarea acestuia în execuție; dacă opțiunea R este prezentă, sînt păstrate toate fișierele de date deschise anterior; în caz contrar, înaintea încărcării programului fis, acestea sînt închise.

Lansarea în execuție dintr-un program BASIC a altui program se poate efectua cu comanda:

```
<B>
CHAIN    [MERGE] "fis" [, [nr1] [, ALL] [, DELETE list]]
```

Aceasta apelează programul fis, îi transmite variabilele din programul curent și îl lansează în execuție începînd de la linia nr1; ALL indică faptul că toate variabilele programului curent vor fi transmise programului apelant (implicit sînt transmise numai cele declarate cu COMMON - vezi capitolul 3.11); MERGE adaugă programul fis la programul din memorie ca segment separat, ce poate fi suprapus peste alt segment cu opțiunea DELETE; DELETE șterge liniile specificate prin lista (lista este de forma prezentată la comanda DELETE).

Observație. Numerele liniilor din lista sînt afectate de comanda RENUM.

Comanda

```
<G>
SHELL    "expsir"
```

încarcă și execută un alt program sau o comandă sistem; expsir este preluată de MS-DOS, prelucrată, după care se revine în GW-BASIC.

● Reluări, trasări

Reluarea execuției unui program întrerupt cu <CTRL>C, respectiv <CTRL><BREAK> sau după execuția unei instrucțiuni STOP sau END se poate efectua cu:

```
<B,G>
CONT
```

Dacă pe parcursul execuției unui program a fost detectată o eroare pentru care utilizatorul dispune de o subrutină de tratare a acesteia, după execuția subrutinei de tratare a erorii, execuția programului poate fi reluată cu comanda

```
<B>
RESUME [0] [NEXT] [nr1]
```

de la linia care a produs eroarea 0, de la următoarea NEXT sau de la cea cu numărul nr1.

Comenzile

```
<B,G>
TRON
<B,G>
TROFF
```

permit activarea (TRON), respectiv dezactivarea (TROFF) 'trasării' execuției programului BASIC prin afișarea numerelor liniilor executate puse între paranteze drepte (' [' și '] ').

● Revenirea în sistemul de operare

Se efectuează cu comanda:

```
SYSTEM
```

care închide toate fișierele deschise și predă controlul acestuia.

Observație. Pe lângă comenzile prezentate, interpretorul GW-BASIC implementat pe microcalculatoare compatibile IBM-PC dispune și de alte comenzi, printre care cele pentru lucrul cu cataloage, ce depășesc însă cadrul prezentării de față.

3.3 Propoziții și fraze (programe) BASIC

Un program BASIC, ca de altfel și în cazul celor scrise în alte limbaje de programare poate fi considerat o frază, în care instrucțiunile reprezintă propoziții. Programul, ca orice frază, trebuie să fie coerent, clar și precis (spre deosebire de exprimarea în limbaj natural, în limbajele de programare nu se admit ambiguități, subînțelesuri, etc.).

Fraza BASIC este o succesiune de linii BASIC, conținând una sau mai multe propoziții (instrucțiuni) încheiată prin STOP sau END.

Liniile BASIC au sintaxa și structura prezentată în capitolul 3.2.

Instrucțiunea STOP are formatul:

```
STOP
```

Ea determină încheierea execuției programului și redă controlul interpretorului, afișând mesajul:

```
Break in 'nrl'
```

unde nrl este numărul liniei ce conține instrucțiunea STOP.

Instrucțiunea

```
END
```

determină, pe lângă încheierea execuției programului, înainte de a reda controlul interpretorului, închiderea tuturor fișierelor deschise. END poate fi plasat oriunde în program.

Observație. Spre deosebire de STOP, END nu determină apariția unui mesaj de sfârșit la consolă.

După efectuarea comenzilor STOP și END execuția programului poate fi reluată din punctul de întrerupere respectiv (dacă după acesta mai există alte instrucțiuni) cu comanda CONT (capitolul 3.2).

Exemplul 3.1

Presupunem că s-a tastat, după inițializarea interpretorului, următoarea secvență:

```
10 PRINT "Primul program BASIC"
20 STOP
```

unde PRINT reprezintă o instrucțiune ce va fi prezentată în capitolul 3.5 și care aici tipărește la display textul cuprins între ghilimele. După tastera comenzii RUN se obține răspunsul:

```
Primul program BASIC
Break in 20
Ok
```

Înlocuind linia 20 cu

```
20 END
```

și retastând RUN, obținem răspunsul:

```
Primul program BASIC
Ok
```

Dacă în loc să folosim două linii (10 și 20) utilizăm una singură, de exemplu:

```
10 PRINT "Primul program BASIC" : STOP
```

după comanda RUN obținem

3. Elemente ale limbajului BASIC

```
Primul program BASIC
Break in 10
Ok
```

În această ultimă variantă, cele două propoziții 'PRINT: Primul program BASIC' și 'STOP' sînt separate prin semnul de punctuație ': '.

Exemplul 3.2

Să extindem puțin programul și să plasăm instrucțiunile STOP și END în interiorul său. Presupunem că am introdus secvența:

```
10 PRINT "Al doilea program BASIC"
20 PRINT "Urmeaza instructiunea STOP"
30 STOP
40 PRINT "Urmeaza instructiunea END"
50 END
60 PRINT "Sfirsitul programului"
70 END
```

în continuare conversația cu calculatorul decurge astfel:

Comanda: RUN
Răspuns: Al doilea program BASIC
Urmează instrucțiunea STOP
Break in 30
Ok

Comanda: CONT
Răspuns: Urmează instrucțiunea END
Ok

Comanda: CONT
Răspuns: Sfîrșitul programului
Ok

Observație. Secvența 10 la 70 prezentată mai sus cuprinde de fapt trei fraze (programe) ce pot fi apelate și executate independent.

De exemplu, dacă se introduce comanda

se obține:

```
RUN 40
Urmează instrucțiunea END
Ok
```

În concluzie o frază BASIC poate conține una sau mai multe propoziții separate prin semnul de punctuație ': ' sau caracterul <CR>. Sfîrșitul frazei este semnalat printr-una din instrucțiunile STOP sau END. Funcție de problema de soluționat ea poate fi mai mult sau mai puțin complexă și, de regulă, conține secvențe de dialog cu utilizatorul.

3.4 Vocabular și elemente de gramatică

Principalele elemente de vocabular ale limbajului BASIC sînt: **setul de caractere, constantele, cuvintele rezervate (cheie) identificatorii și variabilele.**

Gramatica limbajului o constituie mulțimea regulilor semantice și sintactice de formare a propozițiilor corecte BASIC (instrucțiuni, expresii) precum și a frazei (programului).

● Setul de caractere

Reprezintă totalitatea caracterelor ce pot fi utilizate pentru scrierea programelor BASIC. Setul de caractere cuprinde caractere alfabetice (literele mari și mici ale alfabetului latin), caractere numerice (cifrele zecimale 0 la 9) și caractere speciale (+ = - * \ / ^ () % # \$! , ; : ' . & @ _ " <RUBOUT> <ESC> <LF> <CR>).

În BASIC, literele mari și mici, cu excepția celor ce apar în constante și/sau nume de fișier, sînt echivalente.

● Constante

Sînt mărimi introduse direct în program și ale căror valori rămîn neschimbate pe toată durata execuției acestuia. În BASIC există două tipuri de constante: numerice și șiruri.

Constantele numerice reprezintă valori numerice reale ce pot fi exprimate, funcție de tipul lor, sub următoarele forme:

- întregi, cuprinse între -2^{15} și $2^{15}-1$ (-32768 și 32767);
- neîntregi cu punct zecimal: 213.46;
- neîntregi cu exponent: 32E7;
- în dublă precizie: 65589D12;
- hexazecimale (precedate de &H): &H2F4A;
- octale (precedate de &O): &O1777.

Reprezentarea internă a constantelor se poate efectua în două moduri:

- în simplă precizie (dacă are maxim 7 cifre zecimale, este exprimată în forma cu exponentul 'E' sau se termină cu caracterul '!');
- în dublă precizie (dacă are 8 sau mai multe cifre zecimale, este exprimată în forma cu exponentul 'D' sau se termină cu caracterul '#').

Constantele 'sir' sînt secvențe de maxim 255 caractere alfanumerice cuprinse între ghilimele.

● Cuvinte rezervate

Sînt nume simbolice cu semnificații particulare în semantica limbajului. Numărul cuvintelor cheie și semnificațiile lor concrete depind de varianta de BASIC utilizată. În principal cuvintele cheie cuprind numele comenzilor, ale funcțiilor intrinseci precum și cuvintele utilizate în construcția instrucțiunilor.

● Identificatori

Sînt cuvinte utilizate într-un program ca nume de dată și permit descrierea simbolică a acțiunilor în care acestea sînt implicate. Identificatorii mai poartă și denumirea de 'nume simbolice'.

Un identificator începe obligatoriu cu o literă ce poate fi urmată de unul sau mai multe caractere alfabetice, numerice sau de caracterul '.' și trebuie să nu coincidă cu un cuvînt rezervat.

Exemple: A A1R FAM.2 CASA

Observație. Dacă un identificator începe cu literele 'FN' atunci el este nume de funcție definită de utilizator.

● Variabile

Variabilele constituie entități indivizibile în raport cu prelucrările specificate în BASIC. Spre deosebire de constante, valorile variabilelor pot fi modificate pe parcursul execuției programului ce le conține.

Într-un program fiecărei variabile îi corespunde un identificator utilizat în descrierea simbolică a acțiunilor în care acestea apar astfel încît, în reprezentarea operațiilor efectuate cu variabile se utilizează nu valorile, ci numele lor. De exemplu propoziția

$$A = B + C$$

are semnificația: se atribuie variabilei 'A' valoarea sumei variabilelor 'B' și 'C'.

Limbajul BASIC permite lucrul cu variabile numerice (simple sau indexate) și variabile șir.

Variabilele numerice pot fi: întregi, simplă precizie sau dublă precizie, tipul acestora fiind definit printr-un caracter special ce urmează numelui variabilei, astfel:

- '%' variabilă întreagă;
- '!' variabilă simplă precizie;
- '#' variabilă dublă precizie.

Exemple de variabile numerice simple: A! SA% B.C! DRF#.

Numele unei variabile indexate este format dintr-un identificator acceptat de BASIC (eventual însoțit de caracterul special de tip), urmat de una sau mai multe expresii întregi între paranteze rotunde.

Exemple: VEC(6) CF#(I,J) MAT%(K,L+1).

Spre deosebire de variabilele simple, variabilele indexate trebuie declarate înainte de utilizarea cu instrucțiunea neexecutabilă DIM, avînd sintaxa:

DIM ident₁ (n₁₁ [,n₁₂ ...])[ident₂ (n₂₁ [,n₂₂ ...])...]

unde

- ident reprezintă identificatorul variabilei;
- n_{ij} reprezintă valorile maxime ale indicilor.

Instrucțiunea DIM alocă spațiul necesar tablourilor (variabilelor indexate) și inițializează toate componentele cu 0.

Referirea unui element de tablou se efectuează prin identificator și indicii corespunzătorii poziției sale în cadrul acestuia (vezi capitolul 2.4).

Dacă de la un moment dat, pe parcursul execuției unui program, unele variabile indexate nu mai sînt necesare, pentru eliberarea spațiului alocat și a numelor rezervate variabilelor, se poate utiliza instrucțiunea ERASE, cu sintaxa:

```
ERASE ident1 [,ident2 ...]
```

unde ident₁ sînt identificatori de variabile indexate ce figurează într-o instrucțiune DIM anterioară. Dacă ident₁ nu figurează într-o instrucțiune DIM anterioară sau a fost eliberată prin alt ERASE se semnaleză eroare.

Indicii unei variabile indexate iau valori întregi. Valoarea minimă implicită este 0. Această valoare poate fi modificată cu instrucțiunea:

```
OPTION BASE exp
```

care stabilește valoarea minimă a indicelui de tablouri la valoarea expresiei numerice întregi exp, ce poate fi 0 sau 1.

Variabilele de tip șir sînt semnalate prin prezența caracterului 'Ş' imediat după identificatorii acestora.

Exemple de variabile șir: ADŞ VECTŞ SIR1.AŞ.

Remarcă. Dacă un identificator nu este urmat de un caracter special care să-i definească tipul, atunci variabila respectivă este considerată variabilă numerică simplă precizie.

Observație. Utilizarea caracterelor speciale pentru definirea tipurilor variabilelor oferă posibilitatea utilizării aceluiași identificator pentru variabile diferite. Astfel variabilele

```
A% A A# A(I) A%(I) A#(I) AŞ AŞ(I)
```

deși au același identificator (A) sînt distincte.

Pe lângă caracterele speciale, tipurile variabilelor BASIC pot fi definite și după prima literă din identificator prin utilizarea instrucțiunilor DEFINT, DEFNG, DEFDBL și DEFSTR care au sintaxa:

```
DEF { INT
    SNG  l1 [,l2 ...]
    DBL
    STR
```

unde 'l₁ , l₂ , ...' reprezintă o listă de litere.

Tipul variabilelor ai căror identificatori încep cu literele specificate în listă este dat de sufixul utilizat, astfel:

3. Elemente ale limbajului BASIC

INT variabilă întregă;
SNG variabilă simplă precizie;
DBL variabilă dublă precizie;
STR variabilă șir.

Inițializarea variabilelor numerice cu 0 și a variabilelor șir cu caracterul "" (șir nul), se poate efectua global, prin comanda CLEAR cu sintaxa:

```
CLEAR [,mem][,stiva].
```

Utilizată fără parametri, comanda CLEAR, determină inițializarea cu 0 a tuturor variabilelor numerice, cu caracterul "" (NUL) a celor șir, precum și închiderea tuturor fișierelor deschise și eliberarea zonelor tampon atașate acestora.

Utilizată cu parametri, comanda CLEAR, fixează limita memoriei disponibile pentru interpretor la adresa 'mem' și / sau rezervă pentru stivă, 'stiva' octeți.

● Operatori

Sînt simboluri care desemnează funcții de calcul aritmetic, logic sau de comparație a valorilor uneia sau mai multor variabile, pe baza unor date asupra cărora se efectuează operații.

În BASIC există operatori aritmetici, relaționali, logici (prezenți, în ordinea descrescătoare a priorităților, în tabelul 3.1) și funcționali. Ordinea de execuție a operațiilor poate fi modificată prin utilizarea parantezelor rotunde ('(' și ')').

OPERATORI BASIC

Operator ¹⁾	Seminificație	Categorie
^	ridicare la putere	Aritmetici
-	negație (Înmulțire cu -1)	
* /	înmulțire, împărțire	
\ MOD	înmulțire întregă, restul împărțirii întregi	
+ -	adunare, scădere	
< > = => (>=) <= (= <) <> (> <)	mai mic, mai mare egal mai mare sau egal mai mic sau egal diferit	Relaționali

NOT	negație logică	Logici ²⁾
AND	SI logic	
OR	SAU logic	
XOR	sau exclusiv	
IMP	implicație ($X \text{ IMP } Y = 1$ dacă $X \leq Y$)	
EQV	echivalență ($X \text{ EQV } Y = 1$ dacă $X = Y$)	

¹⁾ Operatorii de aceeași prioritate determină execuția operațiilor de la stînga la dreapta

²⁾ Acționează la nivel de bit

Tabelul 3.1

Operatorii aritmetici '`\'`' și '`MOD`' se pot utiliza și cu operanzi neîntregi, dar, în acest caz, înaintea efectuării operației, aceștia sînt convertiți la întregi.

Operatorii relaționali permit compararea a două valori, rezultatul operației fiind adevărat (valoare numerică diferită de zero) sau fals (valoare numerică zero).

Operatorii logici permit efectuarea operațiilor logice la nivel de bit (vezi capitolul 3.6).

Operatorii funcționali sînt nume de funcții intrinseci (predefinite: `SQR` - radical -, `LOG` - logaritm natural -, etc.) sau de funcții definite de utilizator.

● Expresii

Sînt succesiuni de constante și variabile legate prin operatori și, eventual, cuprinse în paranteze.

Funcție de operatorii utilizați, ele pot fi: aritmetice (caz în care rezultatul este o valoare numerică) și logice (rezultatul fiind o valoare de tip boolean: $0 = \text{fals}$ și $\langle \rangle = \text{adevarat}$).

Remarcă. Rezultatul evaluării unei expresii logice poate fi folosit și ca atare (el fiind în fond tot o valoare numerică); a se vedea în acest sens capitolul 3.6.

Dacă într-o expresie intervin variabile sau constante de tipuri diferite, atunci înaintea efectuării operațiilor, operanzii sînt convertiți astfel:

- operatorii aritmetici convertesc toți operanzii la tipul operandului cu precizia cea mai mare, care este și tipul rezultatului expresiei;

- operanzii logici convertesc operatorii în întregi și produc un rezultat întreg.

În cazul în care valoarea expresiei este atribuită unei variabile, atunci ea este convertită la tipul acesteia.

Observații

1. La conversia din virgulă mobilă în întreg, se rotunjește la întregul cel mai apropiat.

2. La conversia din dublă precizie în simplă precizie se rețin primele 7 cifre zecimale, ultima fiind obținută prin rotunjire.

● Comentarii

Pentru a ușura parcurgerea textului și înțelegerea unui program, în textul sursă se pot insera diferite comentarii.

Introducerea comentariilor în textul sursă se poate efectua în două moduri.

O primă modalitate constă în utilizarea instrucțiunii REM cu sintaxa:

```
REM comentariu.
```

Aceasta poate figura oriunde într-un program.

O altă modalitate de introducere a comentariilor este precedarea acestora de caracterul apostrof ('). Comentariile nu sînt luate în considerare la execuția programelor ce le conțin.

Atenție. Dacă se lucrează cu mai multe instrucțiuni pe o linie BASIC, atunci instrucțiunea REM, respectiv comentariul introdus după caracterul apostrof, trebuie să fie ultimele (să precedă caracterul <CR>), în caz contrar toate instrucțiunile sau comenzile aflate între acesta și sfîrșitul liniei (marcat prin caracterul <CR>) sînt considerate comentarii și ca atare, la execuție nu sînt luate în considerare.

3.5 Instrucțiuni de intrare / ieșire

Instrucțiunile de intrare / ieșire permit atribuirea de valori unor variabile utilizînd o listă de constante sau de la tastatură precum și afișarea de valori la display.

Generarea listelor de constante se efectuează cu instrucțiunea neexecutabilă

```
DATA const[,const,...].
```

Ea permite memorarea de constante numerice și șiruri de caractere ce pot fi apoi citite cu instrucțiunea READ. Constantele numerice trebuie separate prin virgulă, iar șirurile de caractere trebuie încadrate în ghilimele ("). Instrucțiunile DATA pot fi plasate oriunde în program. Valorile din lista de constante sînt memorate într-un singur bloc, astfel încît, în cazul cînd sînt mai multe instrucțiuni DATA într-un program, listele lor se concatenează, iar citirea se efectuează în ordinea în care apar față de prima constantă din prima instrucțiune DATA apărută în program.

Instrucțiunea READ are sintaxa

```
READ var[,var,...]
```

și permite atribuirea de valori din lista creată cu instrucțiunea (instrucțiunile) DATA, variabilelor ('var') specificate.

Instrucțiunea READ începe atribuirile cu prima constantă ce urmează celor atribuite printr-un READ anterior, nefiind necesară concordanța între numărul instrucțiunilor DATA și READ. Concordanța se impune însă între tipurile constantelor și al variabilelor cărora le sînt atribuite. Dacă numărul elementelor variabilelor specificate depășește numărul constantelor se semnalează eroare, iar în caz

că numărul lor este mai mic, constantele necitite sînt ignorate.

De exemplu, în urma executării secvenței

```

15 DIM D(2)
20 DATA 5,7.2,"ABC",3,2
.
50 READ A%,B,C$
.
80 DATA 1,3,6.3
.
100 READ D(0),D(1)
.
105 READ E%

```

se vor efectua următoarele atribuiri:

```

A% = 5           B = 7.2           C$ = "ABC"
D(0) = 3         D(1) = 2         E%=1

```

Constantele 3 și 6.3, din lista instrucțiunii DATA de la linia 80 vor fi ignorate.

Exemplul 3.3

Sub rezerva descrierii ulterioare a instrucțiunii PRINT, în listingul 3.1 se exemplifică modul de citire a constantelor dintr-o listă DATA.

```

10 'Citirea constantelor din lista si
15 'afisarea lor la terminal
20 DATA 3.24,7.5,"SIR",8153,0.01,8
25 READ A,B,C$
30 READ D,E,F
35 PRINT "A =",A;"B =",B;"C =",C$
40 PRINT "D =",D;"E =",E;"F =",F
50 STOP
RUN
A =          3.24 B =          7.5 C =          SIR
D =          8153 E =          .01 F =          8
Break in 50
Ok

```

Listing 3.1

Pentru a facilita citirea repetată a unui bloc de dată se utilizează instrucțiunea

RESTORE.

Această determină re poziționarea la începutul blocului de date, astfel încît, primul READ ce urmează unui RESTORE, va începe citirea (atribuirea) de la prima constanta din bloc.

De exemplu secvența

```

10 DATA 1,5.1,6
20 READ A,B
30 RESTORE
40 READ C,D,E

```

3. Elemente ale limbajului BASIC

determină atribuirile:

```
A = 1           B = 5.1
C = 1           D = 5.1           E = 6 .
```

Citirea (introducerea) datelor de la tastatură se realizează cu instrucțiunea

```
INPUT[;] ["text",] var1 [,var2 ...]
```

Execuția acesteia provoacă întreruperea programului și apariția pe ecran a mesajului 'text', dacă acesta a fost specificat. Datele introduse sînt atribuite în ordinea variabilelor din listă. Între tipul variabilelor și valorile introduse trebuie să fie o corespondență biunivocă. De asemenea trebuie să existe concordanță între numărul elementelor variabilelor din listă și valorile introduse. În caz contrar, se semnalează eroare, iar atribuirile nu se efectuează pînă la introducerea corectă.

Recomandare. Pentru a reduce la minim posibilitatea introducerii de date eronate este bine ca fiecare instrucțiune INPUT să fie însoțită de 'text' și să se solicite introducerea unei singure valori.

De exemplu, în locul secvenței:

```
30 INPUT "zi,luna,an :",ZI,LUNA,AN
```

se recomandă

```
30 INPUT "zi   :",ZI
35 INPUT "luna :",LUNA
40 INPUT "an   :",AN
```

Pentru introducerea de la consolă a unei linii (de maxim 255 caractere) se poate utiliza instrucțiunea

```
LINE INPUT[;] ["text" ] vars
```

În urma execuției acesteia, textul introdus (terminat cu caracterul <CR>) este depus în variabila șir 'vars'. 'text' are semnificația prezentată la instrucțiunea INPUT.

Pentru afișarea la display sau la imprimantă a valorilor variabilelor sau a unor anumite situații de ieșire limbajul BASIC dispune de instrucțiunile PRINT și PRINT USING, respectiv LPRINT și LPRINT USING.

Instrucțiunea PRINT are sintaxa

```
PRINT [ ["text"] [exp] [ ["text"] [exp]... ]
```

și permite afișarea pe ecran a valorilor expresiilor din listă într-un format dependent de caracterele utilizate pentru separarea elementelor în listă astfel:

',' - afișarea valorilor următoare se efectuează câte una pe zonă (o zonă = 14 poziții); dacă o variabilă de tipărit depășește lungimea unei zone, elementul următor este tipărit la începutul primei zone libere;

';' sau ' ' - afișarea valorii următoare se efectuează imediat după cea anterioară.

Observație. La afișare, valorile numerice sînt urmate de un spațiu.

Instrucțiunea PRINT USING are sintaxa:

```
PRINT USING sirf;exp1 [,exp2 ...]
```

și permite afișarea valorilor unei liste de expresii numerice sau șir de caractere, conform formatului specificat în 'sirf'. Șirul de formate 'sirf' poate fi un șir de literale anterior definit sau șir de variabile, specificînd formatele cîmpurilor în care urmează a se tipări valorile expresiilor din listă.

Pentru specificarea modului de afișare a cîmpurilor se utilizează unul sau mai multe caractere speciale specifice afișării variabilelor șir, respectiv celor numerice.

Caracterele utilizate în afișarea șirurilor sînt:

- '!' - afișează numai primul caracter din șir;
- '\n_spatii\' - se afișează n+2 caractere din șir; dacă șirul este mai lung, ultimele caractere se neglijează iar dacă este mai scurt, se completează cu zero;
- '&' - cîmp rezervat pentru afișarea unui șir de lungime variabilă; șirul este afișat în întregime;

Exemplul 3.4

Programul prezentat în listingul 3.2 permite introducerea de la tastatură a două șiruri de caractere 'SIR1' și 'SIR2' și reliefează diversele modalități de afișare ale acestora.

```

10 'Citirea si scrierea sirurilor de caractere
15 INPUT "SIR1 = ",A$
20 INPUT "SIR2 = ",B$
25 PRINT "SIR1 = ",A$," SIR2 = ",B$
30 PRINT "SIR1 = ";A$;" SIR2 = ";B$
35 PRINT USING "!" ;A$;" !" ;B$
40 PRINT USING "\ \ " ;A$;B$
45 PRINT USING "&" ;A$;" " ;B$
50 END
RUN
SIR1 = ACESTA ESTE SIRUL UNU
SIR2 = SIRUL DOI
SIR1 = ACESTA ESTE SIRUL UNU SIR2 = SIRUL DOI
A S
  ACESTA SIRUL D
ACESTA ESTE SIRUL UNU SIRUL DOI
Ok
```

Listing 3.2

Caractere utilizate în afișarea numerelor sînt:

3. Elemente ale limbajului BASIC

- '#' - reprezintă poziția fiecărei cifre a numărului;
- '+' - aflat la începutul sau la sfârșitul șirului format, determină afișarea semnelor numerelor (+ sau -) înainte, respectiv în urma lor;
- '' - pus la sfârșitul formatului, determină afișarea semnului pentru numerele negative;
- ''*'' - puse la începutul formatului determină apariția de asteriscuri înaintea unor numerice și oferă posibilitatea afișării unor numere cu două cifre mai lungi decât formatul specificat ;
- '\$\$' - determină apariția unui '\$' la stînga numărului și creează posibilitatea afișării unei cifre în plus față de formatul specificat ;
- ''**\$' - combină efectele caracterelor ''*' și '\$\$';
- ',' - plasat la stînga punctului zecimal, determină apariția unei virgule la fiecare trei cifre în stînga acestuia; plasat înaintea formatului determină apariția unei virgule după fiecare număr afișat;

Observație. Dacă se utilizează semnul exponențial '^^^', virgula nu are efect; '^^^' plasat după pozițiile cifrelor determină afișarea valorii în format exponențial; '' (apostrof) determină afișarea caracterului care îi urmează. Dacă numărul de afișat este mai mare decât timpul specificat prin format, atunci, înaintea numărului, este afișat caracterul '8'.

Exemplul 3.5

În listingul 3.3 se prezintă un program (și rezultatele rulării acestuia) care exemplifică efectul utilizării diferitelor caractere admise în descrierea modului de afișare al variabilelor numerice.

```

5 PRINT "Citirea si scrierea variabilelor numerice"
10 INPUT "N1 = ",N1
15 INPUT "N2 = ",N2#
20 PRINT "N1,N2",N1,N2#
25 PRINT USING "#####.### ";N1;N2#
30 PRINT USING "#####.# ";N1;N2#
35 PRINT USING "+#####.### ";N1;N2#
40 PRINT USING "#####.###- ";N1;N2#
45 PRINT USING "*#####.###- ";N1;N2#
50 PRINT USING "$$#####.###- ";N1;N2#
55 PRINT USING "**$#####.###- ";N1;N2#
60 PRINT USING "**$#####.#- ";N1;N2#
65 PRINT USING "~#####.#- ";N1;N2#
70 PRINT USING "#####^^^", ";N1;N2#
75 PRINT USING "#####.###, ";N1;N2#
80 END
RUN
Citirea si scrierea variabilelor numerice
N1 = 1234
N2 = 987654.321
N1,N2      1234      987654.321
      1234.000      %987654.321
      1234.0      %987654.321
+1234.000      %+987654.321
      1234.000      %987654.321
***1234.000      *987654.321
$1234.000      $987654.321
***$1234.000      *$987654.321

```

```

***$1,234.000    *$987,654.321
~ 1,234.000    ~*987,654.321
1234E+00,    9877D+02,
1234.000,    %987654.321,
Ok

```

Listing 3.3

Pentru afișarea datelor la terminal se mai poate utiliza instrucțiunea:

```
WRITE [exp1 [,exp2 ...]]
```

Spre deosebire de PRINT, aceasta inserează automat <CR><LF> după ultimul element din listă.

Instrucțiunile

LPRINT și LPRINT USING

determină afișarea valorilor variabilelor din listă la imprimantă. Au aceleași sintaxe cu instrucțiunile PRINT, respectiv PRINT USING.

Pentru stabilirea lungimii liniei la display sau imprimantă se utilizează instrucțiunea WIDTH, cu sintaxa:

```
WIDTH [LPRINT] exp
```

unde 'exp' reprezintă o expresie întreagă ce trebuie să aibă valori cuprinse între 0 și 255 și reprezintă numărul maxim de caractere admis pe o linie. Dacă această instrucțiune nu este prezentă, linia se consideră a fi de lungime implicită specifică dispozitivului respectiv.

Observație. Limita inferioară a numărului de caractere, precum și numărul implicit de caractere pe linie diferă funcție de varianta de BASIC utilizată.

Exemplul 3.6

Programul ilustrat în listingul 3.4 prezintă efectul utilizării instrucțiunii WIDTH în două cazuri (cu argument 23, respectiv 40) asupra unui șir de caractere preluat de la tastatură cu instrucțiunea LINE INPUT.

```

10 DATA "Linie de", "caractere"
15 READ L$,C$
20 LINE INPUT "SIR = ",A$
24 N=23
25 WIDTH N
30 PRINT L$;N;C$
35 PRINT A$
39 N=40
40 WIDTH N
45 PRINT L$;N;C$
50 PRINT A$
55 WIDTH 80
60 END
RUN
SIR = Acesta este un sir mai lung de patruzeci de caractere

```

3. Elemente ale limbajului BASIC

```
Linie de 23 caractere
Acesta este un sir mai
lung de patruzeci de ca
ractere
Linie de 40 caractere
Acesta este un sir mai lung de patruzeci
de caractere
Ok
```

Listing 3.4

Observație. Instrucțiunea 'WIDTH 80' din linia 55 are rolul de a restabili lungimea liniei display la 80 caractere; altfel, după execuția programului lungimea liniei ar fi rămas de 40 caractere.

3.6 Operații aritmetice și logice

Permit rezolvarea expresiilor aritmetice și logice, indiferent de forma și complexitatea acestora. Evaluarea expresiilor se efectuează într-o anumită ordine (dată de prioritățile operatorilor utilizați) iar rezultatul, de regulă se atribuie unei variabile. De remarcat faptul că, în urma evaluării unei expresii, ca și în alte limbaje de programare, în BASIC trebuie să avem o singură valoare rezultat (o singură soluție). În cazul cînd în rezolvarea unei probleme pot rezulta mai multe soluții, acestea vor fi de așa natură descrise în BASIC încît fiecare să fie determinată separat.

Instrucțiunea care permite atribuirea valorii rezultate în urma evaluării unei expresii la o variabilă este LET. Această instrucțiune are sintaxa:

```
[LET] var=exp
```

unde var este variabila care primește valoarea rezultată prin evaluarea expresiei exp. Cuvîntul cheie LET este opțional (poate lipsi). Din motive de simplitate și eficiență, în general, acest cuvînt nu se utilizează.

Într-o instrucțiune LET pot apare variabile numerice sau variabile șir (nu simultan).

Variabilele numerice pot fi utilizate în operații aritmetice și logice, iar variabilele șir numai în atribuirii simple și concatenări.

● Operații aritmetice

Adunarea și scăderea se efectuează cu ajutorul operatorilor '+', respectiv '-', avînd semnificația cunoscută din matematică.

Înmulțirea și împărțirea numerelor reale se efectuează utilizînd operatorii '*', respectiv '/'.

Împărțirea întregă a două numere întregi (numerele reale sînt automat convertite în întregi) se efectuează cu operatorul '\'. Evident, în cazul a două numere întregi 'A' și 'B', prime între ele,

```
C1 = A / B   diferă de   C2 = A \ B .
```


De exemplu, în urma execuției secvenței:

```

5 DATA 10,4
10 READ A,B
15 C1=A/B
20 C2=A\B

```

variabilei C1 i se va atribui valoarea '2.5', iar variabilei C2 valoarea '2'.

Restul împărțirii întregi a două numere se obține utilizând operatorul MOD. De exemplu:

```

10 DATA 19,3
20 READ A,B
30 C=A MOD B

```

după evaluarea expresiei din linia 30 variabilei C i se va atribui valoarea '1'.

Cu ajutorul operatorilor '\ ' și 'MOD' putem alcătui o secvență care să descrie împărțirea cu rest:

```

50 CIT% = A% \ B%
60 REST% = A% MOD B%

```

(unde B% <> 0). Evident

$$A\% = CIT\% * B\% + REST\%$$

Observație. La întocmirea și descrierea în BASIC a expresiilor aritmetice trebuie avut în vedere că evaluarea lor să ducă la un număr finit (atenție la împărțirea cu 0) și reprezentabil în calculator. În caz contrar, la execuție, va fi semnalată eroare.

Ridicarea la putere se efectuează cu operatorul '^'.

Operațiile aritmetice se efectuează în ordinea:

- ridicarea la putere (^);
- înmulțirea (*), împărțirea (/), împărțirea întreagă (\) și restul împărțirii întregi (MOD);
- adunarea (+) și scăderea (-).

Dacă este necesar, această ordine poate fi modificată utilizând parantezele '(' și ')'.

Exemplul 3.7

Să se scrie un program BASIC pentru calculul și afișarea valorilor expresiei:

$$E(X,Y) = \frac{1}{2} \frac{X^3}{Y^3} \cdot \frac{X^2+Y^2}{X+Y} + \frac{5-X^2}{X-Y}$$

cu $X > 0$, $Y > 0$ și $X <> Y$.

Programul BASIC și soluția unei execuții sînt prezentate în listingul 3.5.

```

10 PRINT "Program pentru calculul expresiei E(x,y)"
15 INPUT "Introduceti X (X>=0) : ",X
20 INPUT "Introduceti Y (Y>0 SI Y<>X) : ",Y

```

3. Elemente ale limbajului BASIC

```

25 E=(X^(1/3)/Y^(1/4)*(X^(1/2)+Y^2)/(X+Y)+
    (5-X^(1/2))/(X-Y))/2
30 PRINT USING "#####";E
35 END
RUN
Program pentru calculul expresiei E(x,y)
Introduceti X (X>=0) : 6
Introduceti Y (Y>0 SI Y<>X) : 4
18229E-04
Ok

```

Listing 3.5

● Operații logice

Sînt operații care se efectuează bit cu bit. Funcție de tipul variabilelor numerice implicate, numărul de biți cu care se operează poate fi 8, 16, 32 sau 64.

Exemplul 3.8

Fie două numere întregi N% și L%. Utilizînd pe rînd operatorii logici să determinăm valorile variabilei

$$M\% = N\% \text{ 'opl' } L\%$$

unde opl este un operator logic. Dacă luăm $N\% = 26_{(10)} = 11010_{(2)}$ și $L\% = 47_{(10)} = 101111_{(2)}$, atunci pentru M% obținem valorile din tabelul 3.2.

Programul BASIC și rezultatele rulării cu valorile de mai sus sînt prezentate în listingul 3.6.

5	PRINT "Operatii logice"	RUN
10	INPUT "N = ",N%	Operatii logice
15	INPUT "L = ",L%	N = 26
20	M%=NOT N%	L = 47
25	PRINT "NOT N =";M%	NOT N = -27
30	M%=N% AND L%	N AND L = 10
35	PRINT "N AND L =";M%	N OR L = 63
40	M%=N% OR L%	N XOR L = 53
45	PRINT "N OR L =";M%	N IMP L = -17
50	M%=N% XOR L%	N QV L = -54
55	PRINT "N XOR L =";M%	Ok
60	M%=N% IMP L%	
65	PRINT "N IMP L =";M%	
70	M%=N% EQV L%	
75	PRINT "N EQV L =";M%	
80	END	

Listing 3.6

EVALUAREA EXPRESIILOR LOGICE

Operație logică	Evaluare				Exemple ¹⁾ pentru: X=N%=26 ₍₁₀₎ =11010 ₍₂₎ Y=L%=47 ₍₁₀₎ =101111 ₍₂₎
	X	Y	Rezultat		
			Numeric	Logic ²⁾	
NOT X	1	-	0	A	X=0000000000011010 NOT X=1111111111100101 ₍₂₎ =-27 ₍₁₀₎
	0	-	1	F	
X AND Y	1	1	1	F	X=0000000000011010 Y=00000000000101111 X AND Y=000000000001010 ₍₂₎ =10 ₍₁₀₎
	1	0	0	A	
	0	1	0	A	
	0	0	0	A	
X OR Y	1	1	1	F	X=0000000000011010 Y=00000000000101111 X OR Y=0000000000011111 ₍₂₎ =63 ₍₁₀₎
	1	0	1	F	
	0	1	1	F	
	0	0	0	A	
X XOR Y	1	1	0	A	X=0000000000011010 Y=00000000000101111 X XOR Y=0000000000011010 ₍₂₎ =53 ₍₁₀₎
	1	0	1	F	
	0	1	1	F	
	0	0	0	A	
X IMP Y	1	1	1	F	X=0000000000011010 Y=00000000000101111 X IMP Y=1111111111101111 ₍₂₎ =-17 ₍₁₀₎
	1	0	0	A	
	0	1	1	F	
	0	0	1	F	
X EQV Y	1	1	1	F	X=0000000000011010 Y=00000000000101111 X EQV Y=11111111111001010 ₍₂₎ =-54 ₍₁₀₎
	1	0	0	A	
	0	1	0	A	
	0	0	1	F	

¹⁾ Vezi listingul 3.6; ²⁾ A=Adevărat; F=Fals

Tabelul 3.2

● Operații cu variabile șir

Variabilele de tip șir pot apare într-o instrucțiune LET numai în două cazuri:

- în atribuirii simple de forma:

A\$ = B\$

- în concatenări (depunerea valorii unei variabile șir în continuarea valorii alteia) efectuate cu operatorul '+'.

Exemplul 3.9

Concatenarea a două șiruri (listingul 3.7).

```
10 PRINT "Concatenarea a doua siruri"  
15 INPUT "Sirul A : ",A$  
20 INPUT "Sirul B : ",B$  
25 C$=A$+B$  
30 PRINT "Sirul C este : ";  
35 PRINT USING "&";C$  
40 END
```

RUN

Concatenarea a doua siruri

Sirul A : concate

Sirul B : nare

Sirul C este : concatenare

Ok

Listing 3.7

Într-o instrucțiune LET pe lângă operatorii amintiți (aritmetici și logici, în cazul variabilelor numerice, respectiv operatorul '+' <concatenare> în cazul variabilelor șir) mai pot apare și operatori funcționali, desemnând funcții numerice sau șiruri de caractere.

3.7 Funcții și subrutine

Funcțiile și subrutinele sînt seturi de instrucțiuni generalizate, destinate îndeplinirii unei anumite acțiuni, apelabile din diferite puncte ale unui program.

Funcțiile sînt seturi de instrucțiuni apelabile printr-o mnemonică (un nume) și care dau posibilitatea realizării directe a unor operații. Principala caracteristică a funcțiilor este aceea ca în urma evaluării expresiei (expresiilor) pe care o conțin, în program este returnată o singură valoare prin însăși numele ei, astfel încît mnemonicele (identificatorii) lor apar direct în expresii aritmetice sau logice.

Subrutinele constituie module de program care permit realizarea uneia sau mai multor funcțiuni din cele mai diverse. Utilizarea subrutinelor permite realizarea de programe modulare ușor de proiectat și depanat.

Observație. Majoritatea covârșitoare a versiunilor BASIC nu dispune de subrutine (proceduri) în sensul celor întîlnite în alte limbaje de nivel înalt (FORTRAN, PASCAL, C, etc.), ele putînd fi însă, așa cum vom vedea, simulate.

● Funcții predefinite

În această categorie intră o serie de funcții de utilizare frecventă cum sînt funcțiile matematice uzuale: exponențială, logaritmică, valoare absolută, rădăcina pătrată, funcții trigonometrice directe și inverse etc., de intrare / ieșire, funcții referitoare la șiruri, etc.

Principalele funcții predefinite, tipul, semnificația și modul de apel sînt prezentate în tabelul 3.3.

FUNCȚII PREDEFINITE (INTRINSECI)

Funcție	Tip ¹⁾		Rezultat (acțiune)	B	B	G
	Funcție	Argument		A	8	W
				2)	3)	4)
ABS (X)	R	R	Valoarea absolută a lui X	*	*	*
ASC (X\$)	I	S	Codul ASCII al primului caracter din șirul X\$	*	*	*
ATN (X)	R	R	$\arctg(X)$ (în radiani $-\pi/2 < \text{ATN}(X) < \pi/2$)	*	*	*
CDBL (X)	R	R	Conversie simplă, dublă precizie		*	*
CHR\$ (X%)	S	I	Returnează caracterul cu codul ASCII X%	*	*	*
CINT (X)	I	R	Conversie real, întreg	*	*	*
COS (X)	R	R	$\cos(X)$, (X în radiani)	*	*	*
CREAL (X%)	R	I	Conversie întreg, real	*		
CSNG (X)	R	R	Converteste pe X în simplă precizie		*	*
CSRLIN	I	-	Returnează numărul liniei pe care se află cursorul			*
CVI (X\$)	I	S	Converteste un șir de 2 caractere numerice X\$ în valoare întreagă		*	*
CVD (X\$)	R	S	Converteste un șir de 8 caractere numerice X\$ în dublă precizie		*	*
CVS (X\$)	R	S	Converteste un șir de 4 caractere numerice X\$ în simplă precizie		*	*
DATE\$	S	-	Returnează un șir de 10 caractere conținînd data zilei: 11-zz-aaaa			*
DEC\$ (X, F\$)		R,S	Dă o reprezentare zecimală a lui X conform formatului F\$ (vezi PRINT USING)	*		

3. Elemente ale limbajului BASIC

DERR	I	-	Ultimul cod de eroare transmis de sistemul de gestiune al dischetei	*		
EOF (N%)	I	I	Returnează valoarea -1 dacă s-a detectat sfârșitul fișierului de nr. logic N%	*	*	*
ERDEV	I	-	Codul de eroare depistat de ultimul dispozitiv ce a semnalat o eroare			*
ERDEV\$	S	-	Șir de 2 sau 8 caractere conținând numele blocului, respectiv numele dispozitivului caracter ce a semnalat eroarea			*
ERL	I	-	Numărul liniei în care s-a depistat eroarea	*	*	*
ERR	I	-	Codul erorii depistate	*	*	*
EXP (X)	R	R	e^x	*	*	*
FIX (X)	I	R	Partea întreagă a lui X	*	*	*
FRE (0)	I	-	Numărul octeților nefolosiți	*	*	*
HEX\$ (X%)	S	I	Șir reprezentând valoarea hexazecimală a lui X%	*	*	*
HIMEM	I	-	Adresa maximă de memorie utilizată de BASIC	*		
INKEY\$	S	-	Preia un caracter sau un șir nul de la tastatură	*	*	*
INP (X%)	I	I	Returnează octetul citit de la portul X%	*	*	*
INPUT\$ (X%, #Y%)	S	I	Citește X% caractere de la terminalul sau dispozitivul Y%		*	*
INSTR (X%, Y\$, Z\$)	I	I,S	Caută prima apariție a șirului Z\$ în șirul Y\$ începând eventual de la poziția X%	*	*	*
INT (X)	I	R	Returnează partea întreagă a lui X		*	*
IOCTL\$ (#X%)	S	I	Șir de date de control privind modulul de control periferic relativ la dispozitivul de tip caracter cu nr. logic X%			*
JOY (X%)	I	I	Citește starea manetei de joc specificată prin X%	*		
KEY ON			Afișează pe ultima linie a ecranului primele 6 caractere ale valorilor tuturor celor 10 chei			*

LEFT\$ (X\$, Y%)	S	I	Returnează un subsir conținând Y% caractere din șirul X\$	*	*	*
LEN(X\$)	I	S	Returnează lungimea în octeți a lui X\$	*	*	*
LOC(X%)	I	I	Returnează poziția curentă în fișierul cu numărul logic X%		*	*
LOF(X%)	I	I	Returnează lungimea în octeți a fișierului cu numărul logic X%			*
LOG(X)	R	R	$\ln(X)$	*	*	*
LOG10(X)	R	R	$\lg(X)$	*		
LPOS(X%)	I	I	Returnează poziția caracterului curent din zona tampon a imprimantei		*	*
LOWER\$(X\$)	S	S	Schimbă majusculele din X\$ în litere mici	*		
MAX(list)	R	R	Determină valoarea maximă din lista	*		
MID\$(X\$, Y%, Z%)	S	I,S	Returnează un subsir de Z% caractere din X\$, începând cu poziția Y%	*	*	*
MIN(list)	R	R	Determină valoarea minimă din lista	*		
MKD\$(X)	S	R	Conversie dublă precizie, șir de 8 octeți		*	*
MKI\$(X%)	S	I	Conversie întreg, șir de 2 octeți		*	*
MKS\$(X)	S	R	Conversie simplă precizie, șir de 4 octeți		*	*
OCT\$(X)	S	R	Returnează valoarea octală a lui X		*	*
PEEK(X%)	I ⁴	I	Returnează valoarea întreagă a octetului citit din memorie, locația X%	*	*	*
PI	R	-	π	*		
PLAY	I	-	Returnează numărul de note din bufferul melodiei de fond			*
POS(X%)	I	I	Returnează poziția cursorului pe linie	*	*	*
REMAIN(X%)	I	I	Returnează timpul care rămâne de măsurat cu cronometrul X% (0-3) înainte de a-l dezactiva	*		
RIGHT\$(X\$, Y%)	S	I	Returnează un subsir a lui X\$ conținând caracterele de la 1 la Y%	*	*	*

3. Elemente ale limbajului BASIC

RND (X)	R	R	Returnează o valoare pseudoaleatoare în intervalul [0,1]; generatorul de numere aleatoare se inițializează cu instrucțiunea RANDOMIZE	*	*	*
ROUND (X, Y%)	R	R	Rotunjește x la Y% zecimale	*		
SGN (X)	I	R	Returnează: 1 dacă $x > 0$; 0 dacă $x = 0$; -1 dacă $x < 0$.	*	*	*
SIN (X)	R	R	$\sin(X)$; X este exprimat în radiani	*	*	*
SPACE\$ (X%)	S	I	Returnează un șir de X% spații	*	*	*
SPC (X%)	-	I	Produce X% spații în linia de afișat	*	*	*
SQ (X%)	I	I	Indică starea firului de așteptare dintr-un canal sonor dat	*		
SQR (X)	R	R	Radical de ordinul 2 din x	*	*	*
STR\$ (X)	S	R	Returnează un șir de cifre corespunzător valorii lui x	*	*	*
STRING\$ (X%, Y%)	S	I,S	Returnează un șir de X% caractere de codul primului caracter din Y%	*	*	*
TAB (X%)	-	I	Produce spații în linia de afișat pînă în poziția X%	*	*	*
TIME\$	S	-	Returnează un șir de caractere conținînd ora sub forma: hh:mm:ss			*
TAN (X)	R	R	$\text{tg}(X)$; X este exprimat în radiani	*	*	*
USRn (X)	-	R	Activează o funcție scrisă de utilizator în limbaj mașină și îi transmite argumentul X; $n \in [0, 9]$ și este cel definit în DEF USR	*	*	*
UPPER\$ (X%)	S	S	Înlocuiește cu majuscule caracterele alfabetice din X%	*		
VAL (X%)	R	S	Returnează valoarea numerică a șirului de cifre X%	*	*	*
VARPTR (X)	I	-	Returnează adresa relativă a variabilei X sau a unei zone tampon disc		*	*

VARPTR\$(X)	S	-	Returnează adresa relativă a variabilei X sau a unei zone tampon disc, sub forma unui șir de 3 caractere		*	*
XPOS	I	-	Returnează poziția pe axa X a cursorului grafic	*		
YPOS	I	-	Returnează poziția pe axa Y a cursorului grafic	*		

- 1) I=Întreg, R=Real, S=Șir 2) BA=BASIC-AMSTRAD 3) B80=BASIC-80 (MBASIC, GBASIC)
4) GW=GW-BASIC

Tabelul 3.3

Așa cum s-a prezentat în capitolul 3.4 funcțiile constituie operatori și deci apar direct în descrierea în BASIC a unei expresii.

Exemplul 3.10

În listingul 3.8 se prezintă o variantă de program pentru evaluarea expresiei:

$$E(X) = \left[\frac{\sqrt{X}}{3} \cdot \left(\sin\left(Y \cdot \frac{\pi}{180}\right) + \ln(X) \right) \right]$$

cu $X > 0$.

```
10 PRINT "Calculul expresiei E(x,y)"
15 INPUT "X (X>0) : ",X
20 INPUT "Y : ",Y
25 E=INT(SQR(X)/3*(SIN(Y*3.1415/180)+LOG(X)))
30 PRINT "E =";E
35 STOP
```

RUN

Calculul expresiei E(x,y)

X (X>0) : 32.1

Y : 270

E = 4

Break in 35

Ok

Listing 3.8

● Funcții definite de utilizator

Pe lângă funcțiile predefinite, în programele sale, utilizatorul are posibilitatea de a-și defini propriile funcții.

Pentru aceasta are la dispoziție instrucțiunea DEF, cu sintaxa:

3. Elemente ale limbajului BASIC

DEF FNa [(var₁ [,var₂ ...])] = exp

unde FNa reprezintă numele funcției, în care primele două litere sînt obligatoriu FN iar a trebuie să respecte convențiile unui nume de variabilă; var₁ reprezintă argumentele funcției; exp orice expresie BASIC admisă (incluzînd chiar și funcții anterior definite).

O funcție poate fi de tip numeric sau șir și este limitată la expresii ce pot fi scrise pe cel mult o linie.

Definirea unei funcții trebuie efectuată înaintea primei ei utilizări.

Exemplul 3.11

Să se definească o funcție care să permită determinarea suprafeței unui hexagon funcție de raza cercului circumscris acestuia și să se determine cu ajutorul ei suma suprafețelor a două hexagoane. Dacă notăm cu R raza cercului circumscris, atunci suprafața este

$$S = R^2 * \frac{\sqrt{3}}{2}$$

Programul și soluția obținută sînt prezentate în listingul 3.9.

```
10 'Definirea si utilizarea unei functii"
15 DEF FNSH(R)=R^2*SQR(3)/2
20 INPUT "R1 (R1>0) = ",R1
25 INPUT "R2 (R2>0) = ",R2
30 S=FNSH(R1)+FNSH(R2)
35 PRINT "S1 =";FNSH(R1);"S2 =";FNSH(R2);"S =";S
40 STOP
RUN
R1 (R1>0) = 4
R2 (R2>0) = 6
S1 = 13.8564 S2 = 31.1769 S = 45.0333
Break in 40
OK
```

Listing 3.9

● Subrutine BASIC

În general, subrutinele sînt utilizate în cazul cînd o secvență de instrucțiuni este folosită de mai multe ori într-un program, pentru a elimina necesitatea repetării acestora.

O utilizare însă mult mai eficientă a subrutinelor se obține atunci cînd, pentru o clasă de probleme se construiesc (sau sînt constituite) biblioteci de subrutine. În acest caz, realizarea programelor (așa cum se va vedea în capitolele următoare) se poate simplifica într-atît încît să se ajungă ca acesta să fie alcătuit doar din apeluri la subrutine deja definite și testate.

În lucrul cu subrutine se utilizează două instrucțiuni: GOSUB și RETURN.

Instrucțiunea GOSUB are sintaxa:

GOSUB etich

unde etich reprezintă adresa primei linii a subrutinei.

În momentul înlînirii acestei instrucțiuni, programul este întrerupt, se reține adresa următoarei

instrucțiuni și se face salt la linia cu numărul etich, execuția continuând cu prima instrucțiune executabilă de după aceasta pînă în momentul întâlnirii instrucțiunii RETURN.

Instrucțiunea RETURN are sintaxa:

RETURN

și determină revenirea la adresa instrucțiunii imediat următoare ultimei instrucțiuni GOSUB executate.

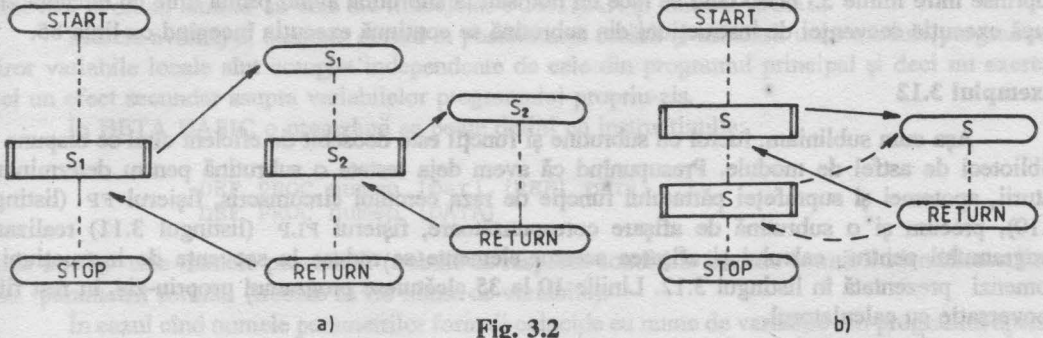


Fig. 3.2

În figura 3.2 este prezentat, utilizând schemele logice, modul de apel și revenirea din subrutine în configurațiile acceptate. Descrierea în BASIC a apelurilor și revenirilor din subrutine este ilustrată în figura 3.3.

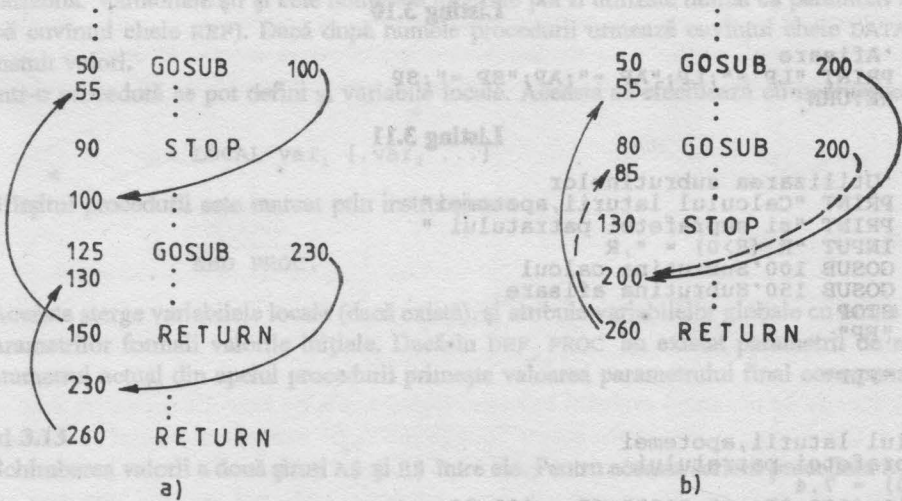


Fig. 3.3

În cazul secvențelor prezentate în figura 3.2a execuția decurge astfel: se execută instrucțiunile anterioare liniei 50; la întâlnirea instrucțiunii GOSUB 100 din linia 50 se sare la linia 100 execuția continuând de aici pînă la linia 125, cînd se face un nou salt, de data aceasta la subrutina a cărei

3. Elemente ale limbajului BASIC

primă linie are eticheta 230; după parcurgerea liniilor 230 la 260, la întâlnirea instrucțiunii RETURN (linia 260), se revine la instrucțiunea aflată imediat după ultimul salt la subrutina (efectuat la linia 125), respectiv la linia 130; se parcurg liniile 130 la 150, după care se revine, de data aceasta, la linia 55.

În secvența prezentată în figura 3.2b se execută instrucțiunile pînă la linia 50, se sare la linia 200 (salt la subrutină) se parcurge subrutina și se revine la linia 55; se execută apoi instrucțiunile cuprinse între liniile 55 la 80 cînd se face un nou salt la subrutina avînd prima linie cu eticheta 200; după execuția secvenței de instrucțiuni din subrutină se continuă execuția începînd cu linia 85.

Exemplul 3.12

Așa cum subliniam, lucrul cu subrutine și funcții este deosebit de eficient cînd se dispune de biblioteci de astfel de module. Presupunînd că avem deja testată o subrutină pentru determinarea laturii, apotemei și suprafeței pătratului funcție de raza cercului circumscris, fișierul FP (listingul 3.10), precum și o subrutină de afișare corespunzătoare, fișierul FLP (listingul 3.11) realizarea programului pentru calculul și afișarea acestor elemente se reduce la secvența de instrucțiuni și comenzi prezentată în listingul 3.12. Liniile 10 la 35 alcătuiesc programul propriu-zis, în rest fiind conversație cu calculatorul.

```
100 'Calcul LP,AP,SP
105 LP=R*SQR(2)
110 AP=R*SQR(2)/2
115 SP=2*R^2
120 RETURN
```

Listing 3.10

```
150 'Afișare
155 PRINT "LP =";LP;"AP =";AP;"SP =";SP
160 RETURN
```

Listing 3.11

```
10 'Utilizarea subrutinelor
15 PRINT "Calculul laturii,apotemei"
16 PRINT "si suprafeței patratului "
20 INPUT "R (R>0) = ",R
25 GOSUB 100'Subrutina calcul
30 GOSUB 150'Subrutina afișare
35 STOP
```

```
MERGE "FP"
Ok
MERGE "FLP"
Ok
RUN
Calculul laturii,apotemei
si suprafeței patratului
R (R>0) = 7.4
LP = 10.4652 AP = 5.23259 SP = 109.52
Break in 35
Ok
```

Listing 3.12

Remarcă. În majoritatea variantelor BASIC, lucrul cu subrutine are un mare neajuns: variabilele utilizate în interiorul acestora, așa numiții parametrii formali, sînt de fapt globale, adică,

deși sînt definite în subrutină, ele se regăsesc și în programul apelant, ceea ce creează dificultăți în apelul repetat, cu diferiți parametri actuali, ai aceleiași subrutine. În capitolele 4 și 5 sînt prezentate exemple privitoare la modul de înlăturare, pe cît posibil, a acestui inconvenient.

● Subrutine BETA BASIC

Subliniem totuși că varianta BETA BASIC permite lucrul cu subrutine (proceduri) la un nivel mult mai evoluat, similar cu cele ale limbajelor FORTRAN și chiar PASCAL.

Marele avantaj al acestora constă în posibilitatea creării și utilizării unor module program ale căror variabile locale sînt complet independente de cele din programul principal și deci nu exercită nici un efect secundar asupra variabilelor programului propriu-zis.

În BETA BASIC o procedură se poate defini cu instrucțiunile:

```
DEF PROC numep [par[, [REF] par]...]
DEF PROC numep [DATA]
```

unde numep este numele procedurii (trebuie să respecte condițiile impuse formării identificatorilor) par parametrii formali (trebuie să fie nume de variabile).

În cazul cînd numele parametrilor formali coincide cu nume de variabile din programul apelant atunci, pe perioada execuției procedurii, aceștia sînt salvați iar după revenirea din procedură, restaurează astfel încît valorile lor nu sînt afectate.

Dacă înaintea unui parametru formal apare însă cuvîntul REF, atunci, la sfîrșitul procedurii valoarea parametrului formal respectiv este atribuită parametrului actual corespunzător care trebuie să fie o variabilă. Variabilele șir și cele numerice indexate pot fi utilizate numai ca parametri referință (deci după cuvîntul cheie REF). Dacă după numele procedurii urmează cuvîntul cheie DATA atunci nu se transmit valori.

Într-o procedură se pot defini și variabile locale. Aceasta se efectuează cu instrucțiunea

```
LOCAL var1 [, var2 ...]
```

Sfîrșitul procedurii este marcat prin instrucțiunea

```
END PROC.
```

Aceasta șterge variabilele locale (dacă există), și atribuie variabilelor globale cu nume identice cu ale parametrilor formali valorile inițiale. Dacă în DEF PROC au existat parametri de referință, atunci parametrul actual din apelul procedurii primește valoarea parametrului final corespunzător.

Exemplul 3.13

Schimbarea valorii a două șiruri A\$ și B\$ între ele. Pentru aceasta scriem procedura SCHIMBA:

```
1000 DEF PROC SCHIMBA REF A$, REF B$
1010 LOCAL AUX$
1020 AUX$=A$
1030 A$=B$
1040 B$=AUX$
1050 END PROC
```

3. Elemente ale limbajului BASIC

Am folosit variabila auxiliară AUX\$ pe care am definit-o ca locală și variabilele A\$ și B\$ care constituie parametri formali ai procedurii.

Apelul procedurii într-un program se efectuează prin enunțarea numelui ei urmat de cele ale parametrilor actuali.

Dacă înaintea execuției secvenței:

```
10 DATA "SIRA", "SIRB", "SIRC"  
20 READ A$, B$, C$  
30 SCHIMBA A$, B$  
40 SCHIMBA B$, C$
```

A\$ = SIRA, B\$ = SIRB, C\$ = SIRC, după execuția sa variabilele A\$, B\$ și C\$ vor conține valorile:

A\$ = SIRB, B\$ = SIRC respectiv C\$ = SIRA.

3.8 Instrucțiuni de control

Limbajul BASIC cuprinde un set de instrucțiuni de control care acoperă o bună parte din structurile de control cu care operează programarea structurată, precum și instrucțiuni de control nestructurate.

În categoria instrucțiunilor de control structurate intră pe lângă instrucțiunile pentru structuri liniare, cele destinate descrierii structurilor alternative și repetitive.

Instrucțiunile de control nestructurate cuprind în principal salturile condiționate și necondiționate. Totuși, utilizate cu atenție, acestea pot servi la crearea structurilor prezentate în capitolul 2.3, cu atât mai mult cu cât, din acest punct de vedere limbajul BASIC este deficitar.

● Instrucțiuni alternative

Sînt de formele:

```
IF expl THEN ina [ELSE inf]  
IF expl THEN nrla [ELSE nrlf]  
IF expl GOTO nrla [ELSE nrlf]
```

și permit, funcție de rezultatul evaluării expresiei logice expl, fie execuția secvenței ina sau salt la linia nrla (dacă expl are valoarea logică 'adeverat'), fie execuția secvenței inf, respectiv salt la linia nrlf (dacă expl are valoarea logică 'fals'), cînd ramura ELSE este prezentă.

Într-o instrucțiune IF, secvențele ina, respectiv inf pot conține una sau mai multe instrucțiuni separate prin caracterul ':', fără a depăși însă lungimea maximă a unei linii.

Deoarece, în majoritatea variantelor BASIC, nu există un cuvînt cheie care să semnaleze sfîrșitul unei instrucțiuni IF, ci doar caracterul <CR>, care semnalează și sfîrșitul liniei, pe aceeași linie BASIC, nu pot fi scrise după aceasta alte instrucțiuni. În cazul cînd acestea există, ele sînt considerate incluse în IF.

De exemplu, în secvența

```
60 IF A = 0 THEN B=B+1 ELSE B=0 : C=C+2 <CR>
```

dacă expresia $A=0$ este adevărată se execută atribuirea $B=B+1$ și se trece la linia următoare, în caz contrar, se execută instrucțiunile $B=0$ și $C=C+2$.

Observație. Caracterul ' : ' utilizat aici are rolul de a separa cele două instrucțiuni conținute în ramura ELSE și nu de a delimita instrucțiunea

```
IF A=0 THEN B=B+1 ELSE B=0 de instrucțiunea C=C+2.
```

În BASIC instrucțiunile alternative trebuie deci considerate astfel:

```
IF expl THEN instr ELSE instr <CR>.
```

Exemplul 3.14

Să scriem un program care să determine maximumul din trei valori numerice.

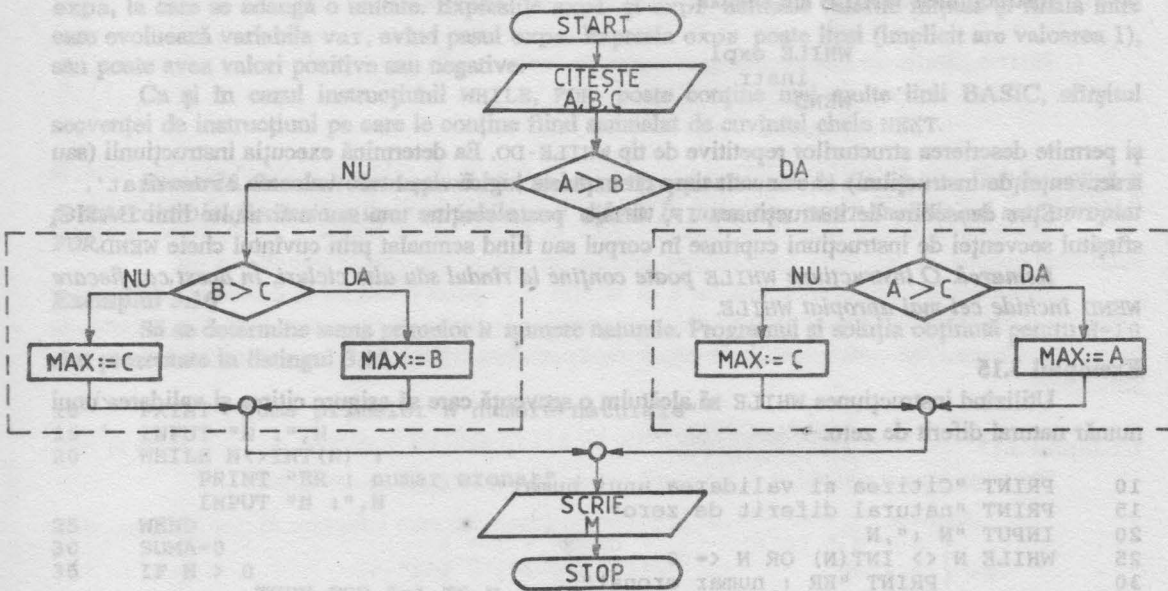


Fig. 3.4

Schema logică este cea din figura 3.4, iar programul și soluțiile obținute sînt prezentate în listingul 3.13.

3. Elemente ale limbajului BASIC

```
10 PRINT "Maxim din 3 valori"      RUN
15 INPUT "A : ",A                 Maxim din 3 valori
20 INPUT "B : ",B                 A : 3
25 INPUT "C : ",C                 B : 5
30 IF A>B                          C : 6
    THEN IF A>C                     MAX = 6
        THEN MAX=A                   Ok
        ELSE MAX=C
    ELSE IF B>C
        THEN MAX=B
        ELSE MAX=C
35 PRINT "MAX =" ;MAX
40 END
```

Listing 3.13

● Instrucțiuni repetitive

Din această categorie limbajul BASIC dispune de instrucțiunile WHILE și FOR.

Instrucțiunea WHILE are sintaxa:

```
WHILE expl
  instr
WEND
```

și permite descrierea structurilor repetitive de tip WHILE-DO. Ea determină execuția instrucțiunii (sau a secvenței de instrucțiuni) instr atât timp cât expresia logică expl are valoarea 'adeverat'.

Spre deosebire de instrucțiunea IF, WHILE poate conține una sau mai multe linii BASIC, sfârșitul secvenței de instrucțiuni cuprinse în corpul sau fiind semnalat prin cuvântul cheie WEND.

Remarcă. O instrucțiune WHILE poate conține la rândul său alte cicluri. În acest caz fiecare WEND închide cel mai apropiat WHILE.

Exemplul 3.15

Utilizând instrucțiunea WHILE să alcătuim o secvență care să asigure citirea și validarea unui număr natural diferit de zero.

```
10 PRINT "Citirea si validarea unui numar"
15 PRINT "natural diferit de zero"
20 INPUT "N : ",N
25 WHILE N <> INT(N) OR N <= 0
30   PRINT "ER : numar eronat"
35   INPUT "N : ",N
40 WEND
45 PRINT "N =" ;N ; " valoare corecta"
50 END
```

RUN

Citirea si validarea unui numar

natural diferit de zero

N :-1

ER : numar eronat

N :2.3


```

ER : numar eronat
N :5
N = 5  valoare corecta
Ok

```

Listing 3.14

Secvența de program și testarea ei sînt prezentate în listingul 3.14. Instrucțiunea WHILE cuprinde liniile 25 la 40, conținînd în corpul său alte două instrucțiuni: PRINT (linia 30), respectiv INPUT (linia 35). Aceasta din urmă este necesară pentru a modifica, dacă se dorește, valoarea expresiei logice din linia 25; altfel o dată intrat în WHILE, programul ciclează. Dacă N este corect (este întreg și mai mare decît zero) liniile 25 la 40 nu sînt executate niciodată.

Instrucțiunea FOR are sintaxa:

```

FOR var = expi TO expf [STEP exps]
  instr
NEXT [var]

```

și permite descrierea în BASIC a structurilor repetitive cu număr cunoscut de pași (vezi capitolul 2.3). Numărul de ori de cît se execută instrucțiunea (secvența de instrucțiuni) instr este dat de cîtul împărțirii întregi a diferenței dintre valorile expresiilor numerice expf și expi, la aceea a expresiei exps, la care se adaugă o unitate. Expresiile expi și expf definesc valorile inițială și finală între care evoluează variabila var, avînd pasul exps. Expresia exps poate lipsi (implicit are valoarea 1), sau poate avea valori pozitive sau negative.

Ca și în cazul instrucțiunii WHILE, FOR poate conține mai multe linii BASIC, sfîrșitul secvenței de instrucțiuni pe care le conține fiind semnalat de cuvîntul cheie NEXT.

Remarcă. Se admit instrucțiuni FOR imbricate (conținute unele în altele), cu condiția utilizării pentru controlul fiecăreia a unor variabile var diferite. În acest caz, NEXT închide cel mai apropiat FOR.

Exemplul 3.16

Să se determine suma primelor N numere naturale. Programul și soluția obținută pentru N=10 sînt prezentate în listingul 3.15.

```

10 PRINT "Suma primelor N numere naturale"
15 INPUT "N : ",N
20 WHILE N<>INT(N) :
    PRINT "ER : numar eronat" :
    INPUT "N : ",N
25 WEND
30 SUMA=0
35 IF N > 0
    THEN FOR I=1 TO N :
        SUMA=SUMA+I :
    NEXT I
40 PRINT "Suma = ";SUMA
45 END
RUN
Suma primelor N numere naturale
N :10
Suma = 55
Ok

```

Listing 3.15

3. Elemente ale limbajului BASIC

Suma se efectuează (linia 35) numai dacă $N > 0$. Valoarea acestuia este validată de instrucțiunea WHILE din linia 20.

● Instrucțiuni de salt

Pot fi de salt necondiționat sau condiționat.

Instrucțiunea de salt necondiționat este:

```
GOTO nrlin
```

și determină saltul la linia cu numărul nrlin. La întâlnirea acestei instrucțiuni execuția programului continuă cu linia nrlin, dacă este o instrucțiune executabilă, sau cu prima instrucțiune executabilă de după linia specificată.

Remarcă. Deși, de multe ori, dintr-o aparentă simplitate, există tendința de a folosi această instrucțiune în controlul programelor, totuși, pentru realizarea unor programe ușor de urmărit, de depanat și, de ce nu, elegante, se recomandă evitarea utilizării ei, prin descrierea cu ajutorul instrucțiunilor structurate a eventualelor ramificații ale programului.

Instrucțiunile de salt necondiționat sînt de forma:

```
ON exp GOTO nrlin1 [,nrlin2 ...]  
ON exp GOSUB nrlin1 [,nrlin2 ...]
```

și presupun mai întii evaluarea expresiei întregi exp și, funcție de valoarea acesteia, salt la prima, la a doua sau a n-a valoare nrlin din listă (semnificînd numărul liniei, respectiv salt la subrutină).

Deosebirea dintre cele două forme constă în faptul că în forma a doua, după execuția subrutinei apelate, se revine la instrucțiunea imediat următoare, pe cînd în primul caz nu.

Saltul condiționat la subrutina ON exp GOSUB ... poate fi utilizat la descrierea structurilor alternative generalizate (capitolul 2.3). Datorită revenirii la linia imediat următoare, determinată de lucrul cu subrutine, această instrucțiune are de fapt o singură intrare și o singură ieșire, și deci, este structurată.

Exemplul 3.17

Fie N un număr natural și A, B două numere reale date. Să se determine:

$$F = \begin{cases} N + B * \text{Ln}(N) & \text{dacă } N = 3 * k \\ A * \sqrt{N} & \text{dacă } N = 3 * k + 1 \\ A / B * N & \text{dacă } N = 3 * k + 2 \text{ cu } k \end{cases}$$

Pentru aceasta vom utiliza instrucțiunea ON exp GOSUB ..., unde exp este $N \text{ MOD } 3 + 1$, luînd deci valori în mulțimea {1,2,3}. Programul sursă și soluțiile obținute sînt prezentate în listingul 3.16.

```
10 PRINT "Calculul valorii functiei F"  
15 DATA 1.5,.5  
20 READ A,B
```

```

30 INPUT "N :",N
35 WHILE N<>INT(N) OR N<0 :
    PRINT "ER : numar eronat" :
    INPUT "N =",N :
    WEND
40 ON N MOD 3 + 1 GOSUB 60,70,80
45 PRINT "F =",F
50 END
60 'Subrutina 1
65 F=N+B*LOG(N) : RETURN
70 'Subrutina 2
75 F=A*SQR(N) : RETURN
80 'Subrutina 3
85 F=A/B*N : RETURN
RUN

```

Calculul valorii functiei F

```

N :6
F = 6.89588
Ok

```

```

RUN
Calculul valorii functiei F
N :7

```

```

F = 3.96863
Ok

```

```

RUN
Calculul valorii functiei F
N :8

```

```

F = 24
Ok

```

Listing 3.16

0 instrucțiune utilă, de salt condiționat, este și:

```
ON ERROR GOTO nrlin
```

care determină, în caz de eroare, salt la o rutină de tratare a erorilor scrisă de utilizator.

În diferite variante de BASIC întâlnim și alte instrucțiuni de salt condiționat, cum sînt:

```
ON KEY nr GOSUB nrlin(*)
```

salt la subrutina care debutează cu linia nrlin la acționarea tastei funcționale nr;

```
ON TIMER nr GOSUB nrlin(*)
```

întrerupere și salt la subrutina care debutează cu linia nrlin la fiecare nr secunde;

```
ON PLAY nr GOSUB nrlin(*)
```

salt la subrutina specificată prin nrlin de tratare a evenimentului provocat de diminuarea numărului de note din zona tampon de melodii sub nr note;

```
ON COM nr GOSUB nrlin(*)
```

salt la subrutina specificată prin nrlin de tratare a evenimentului provocat de sosirea caracterelor în zona tampon de comunicații a canalului nr;

3. Elemente ale limbajului BASIC

```
ON BREAK CONT(**)
```

anulează acțiunea tastei <ESC>, împiedicând oprirea execuției programului;

```
ON BREAK GOSUB nrlin(**)
```

salt la subrutina specificată prin nrlin, la apăsarea de două ori a tastei <ESC>;

```
ON BREAK STOP(**)
```

determină oprirea programului la acționarea tastei <ESC>.

Pentru descrierea cât mai sugestivă și mai firească a unui algoritm utilizând limbajul BASIC, instrucțiunile de control pot fi combinate între ele (în sensul că pot fi conținute unele în altele), în cadrul uneia sau mai multor linii BASIC. O astfel de utilizare trebuie efectuată însă cu multă atenție și cu respectarea riguroasă a modului de constituire și execuție a instrucțiunilor. În caz contrar, deși aparent corect, programul poate furniza soluții eronate.

Exemplul 3.18

Să se alcătuiască un program pentru determinarea valorii maxime și minime dintr-un tablou unidimensional A() de 6 elemente. O posibilă soluționare a problemei este ilustrată în schema logică din figura 3.5.

Întocmim după aceasta programul BASIC prezentat în listingul 3.17. Lansat în execuție acesta nu dă întotdeauna soluții corecte; astfel dacă variabila indexată A() conține șirul de valori 1, 0, 3, 4, 5, 6, soluția este eronată. Aceasta se datorează faptului că, dacă pentru I=1, în urma evaluării expresiei A(I)>MAX(0>1) rezultă, pentru aceasta, valoarea logică 'fals', efectuându-se instrucțiunile de pe ramura ELSE (unde cum 0 < 1, => MIN=0), inclusiv NEXT, pentru I=2, expresia A(2)>MAX(3>1), fiind adevărată, se efectuează atribuirea MAX=3 și, înlăturându-se cuvântul ELSE, se iese din linia BASIC 50 (sărindu-se de fapt peste NEXT).

```
10 'Un program gresit
15 DIM A(5)
20 FOR I=0 TO 5
25 PRINT "A(";I;")";
30 INPUT " = ",A(I)
35 NEXT
40 MAX=A(0)
45 MIN=A(0)
50 FOR I=1 TO 5 :
    IF A(I)>MAX
        THEN MAX=A(I)
        ELSE IF A(I)<MIN
            THEN MIN=A(I) :
    NEXT
55 PRINT "MAX = ";MAX;"MIN = ";MIN
60 END
```

```
RUN
A( 0 ) =1
A( 1 ) =0
A( 2 ) =3
A( 3 ) =4
A( 4 ) =5
A( 5 ) =6
MAX = 3 MIN = 0
Ok
```

Listing 3.17

```
10 PRINT "Calculul valorii minime și maxime ale funcției f(x) = x^2 - 2x + 1"
15 DATA 1, 5, 5
20 READ A, B
```

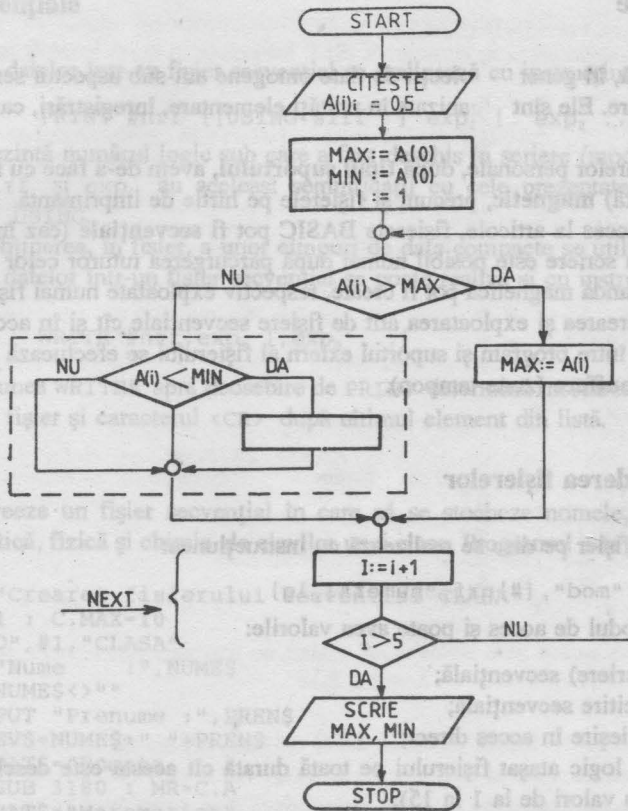


Fig. 3.5

Remediere. Se scrie instrucțiunea NEXT pe o linie BASIC separată. Programul sursă corect și execuția acestuia sînt prezentate în listingul 3.18.

```

10 'Programul corect
15 DIM A(5)
20 FOR I=0 TO 5
25 PRINT "A(";I;")";
30 INPUT " = ",A(I)
35 NEXT
40 MAX=A(0)
45 MIN=A(0)
50 FOR I=1 TO 5 :
    IF A(I)>MAX
    THEN MAX=A(I)
    ELSE IF A(I)<MIN
    THEN MIN=A(I)
55 PRINT "MAX = ";MAX;"MIN = ";MIN
60 END
    RUN
    A ( 0 ) = 1
    A ( 1 ) = 0
    A ( 2 ) = 3
    A ( 3 ) = 4
    A ( 4 ) = 5
    A ( 5 ) = 6
    MAX = 6 MIN = 0
    Ok
51 NEXT
    
```

Listing 3.18

3.9 Lucrul cu fișiere

Fișierele reprezintă, în general colecții de date omogene atît sub aspectul semnificației cît și al necesităților de prelucrare. Ele sînt organizate în unități elementare, înregistrări, care, la nivel logic se numesc articole.

În cazul calculatoarelor personale, după tipul suportului, avem de-a face cu fișiere pe bandă (casetă) și pe disc (dischetă) magnetic, precum și fișierele pe hîrtie de imprimantă.

După modul de acces la articole, fișierele BASIC pot fi secvențiale (caz în care accesul la un articol pentru citire sau scriere este posibil numai după parcurgerea tuturor celor anterioare) și în acces direct. Pe / de pe, bandă magnetică pot fi create, respectiv exploatate numai fișiere secvențiale. Discul magnetic permite crearea și exploatarea atît de fișiere secvențiale cît și în acces direct.

Transferul datelor între program și suportul extern al fișierului se efectuează cu ajutorul unor zone de memorie numite buffere (zone tampon).

● Deschiderea și închiderea fișierelor

Deschiderea unui fișier pe disc se realizează cu instrucțiunea:

```
OPEN "mod", [#]nrf, "numef" [,lg]
```

unde: "mod" specifică modul de acces și poate avea valorile:

- O - ieșire (scriere) secvențială;
- I - intrare (citire) secvențială;
- R - intrare / ieșire în acces direct;
- nrf - numărul logic atașat fișierului pe toată durata cît acesta este deschis (expresie întreagă cu valori de la 1 la 15);
- numef - numele fișierului;
- lg - specifică lungimea în octeți a înregistrării în cazul fișierelor cu acces direct.

Observație. Se admite deschiderea simultană a aceluiași fișier sub mai multe numere nrf, pentru citirea secvențială sau în acces direct, dar numai cu n singur număr pentru scriere secvențială.

Închiderea fișierelor pe disc se realizează cu instrucțiunea:

```
CLOSE [#]nrf1 [, [#]nrf2 ...]
```

unde: nrf₁ numerele logice sub care au fost deschise fișierele respective.

După închiderea unui fișier nrf poate fi utilizat la deschiderea aceluiași sau a altui fișier.

Instrucțiunea CLOSE fără argumente (parametri) determină închiderea tuturor fișierelor deschise.

Observație. Instrucțiunea END și comanda NEW închid automat toate fișierele deschise. Instrucțiunea STOP nu închide fișiere.

● Fişiere secvenţiale

Scrierea datelor într-un fişier secvenţial se realizează cu instrucţiunea:

```
PRINT #nrf [[USING sirf ] exp1 [ exp2 ...]]
```

unde: nrf reprezintă numărul logic sub care a fost deschis în scriere (mod "O") fişierul.

Şirul sirf şi exp₁ au aceleaşi semnificaţii cu cele prezentate la instrucţiunile PRINT, respectiv PRINT USING.

Pentru obţinerea, în fişier, a unor cimpuri de date compacte se utilizează caracterul ' ; '.

Scrierea datelor într-un fişier secvenţial se poate realiza şi cu instrucţiunea:

```
WRITE #nrf,exp1 [,exp2 ...]
```

Instrucţiunea WRITE# spre deosebire de PRINT determină inserarea unei virgule după fiecare valoare scrisă în fişier şi caracterul <CR> după ultimul element din listă.

Exemplu 3.19

Să se creeze un fişier secvenţial în care să se stocheze numele, prenumele şi mediile la română, matematică, fizică şi chimie ale elevilor unei clase. Programul este prezentat în listingul 3.19.

```

5 PRINT "Crearea fisierului secvential CLASA"
10 C.MIN=1 : C.MAX=10
15 OPEN "O",#1,"CLASA"
20 INPUT "Nume      :",NUME$
25 WHILE NUME$<>""
30     INPUT "Prenume :",PREN$
35     ELEV$=NUME$+" "+PREN$
40     C.MAT$="Romana   "
45     GOSUB 3180 : MR=C.A
50     C.MAT$="Matematica"
55     GOSUB 3180 : MM=C.A
60     C.MAT$="Fizica   ."
65     GOSUB 3180 : MF=C.A
70     C.MAT$="Chimie   "
75     GOSUB 3180 : MC=C.A
80     PRINT ELEV$;" " : " ;MR;MM;MF;MC
85     INPUT "Articol corect ? (Y/N) : ",C$
90     IF C$ = "Y" OR C$ = "y"
95         THEN WRITE #1,ELEV$,MR,MM,MF,MC
          INPUT "Nume      :",NUME$
100 WEND
105 END
3180 'Citeste si valideaza un numar intreg
3181 'cuprins intre C.MIN si C.MAX
3185 PRINT C.MAT$;"(" ;C.MIN;"<= N <=" ;C.MAX;")";
3190 INPUT " = ",C.A
3195 WHILE C.A<C.MIN OR C.A>C.MAX OR C.A<>INT(C.A) :
      PRINT "ER : valoare eronata" :
      PRINT C.MAT$;"(" ;C.MIN;"<= N <=" ;C.MAX;")";
      INPUT " = ",C.A :
WEND
3200 RETURN

```

3. Elemente ale limbajului BASIC

```
RUN
Crearea fisierului secvential CLASA
Numele      :Alexandrescu
Prenumele   :Vasile
Romana      ( 1 <= N <= 10 ) = 9
Matematica  ( 1 <= N <= 10 ) = 10
Fizica      ( 1 <= N <= 10 ) = 10
Chimie      ( 1 <= N <= 10 ) = 9
Alexandrescu Vasile : 9 10 10 9
Articol corect ? (Y/N) : Y
Numele      :Florea
Prenumele   :Doina
Romana      ( 1 <= N <= 10 ) = 9
Matematica  ( 1 <= N <= 10 ) = 8
Fizica      ( 1 <= N <= 10 ) = 9
Chimie      ( 1 <= N <= 10 ) = 7
Florea Doina : 9 8 9 7
Articol corect ? (Y/N) : Y
Numele      :Radulescu
Prenumele   :Luminita
Romana      ( 1 <= N <= 10 ) = 10
Matematica  ( 1 <= N <= 10 ) = 9
Fizica      ( 1 <= N <= 10 ) = 9
Chimie      ( 1 <= N <= 10 ) = 9
Radulescu Luminita : 10 9 9 9
Articol corect ? (Y/N) : Y
Numele      :
Ok
```

Listing 3.19

Subrutina cuprinsă între liniile 3180 la 3200 asigură citirea și validarea mediilor (valori rotunjite); acestea trebuie să fie numere întregi și să se încadreze între 1 (C.MIN) și 10 (C.MAX), valori precizate în linia 10. Articolul este scris în fișier numai dacă este considerat corect (liniile 85 și 90).

Observație. Deoarece închiderea fișierului se efectuează la întâlnirea instrucțiunii END, nu mai este necesară, în acest caz, scrierea explicită a unui CLOSE.

Citirea datelor dintr-un fișier secvențial se realizează cu instrucțiunea:

```
INPUT #nrf,var1 [,var2 ...]
```

unde: nrf numărul logic sub care a fost deschis în citire fișierul (mod "I"); var₁ lista variabilelor care primesc valori în ordinea specificată în listă.

Observație. Între tipul și ordinea acestor variabile și reprezentarea lor în fișier trebuie să fie o corespondență biunivocă.

Atribuirile se fac astfel: primul caracter din fișier diferit de <CR>, spațiu () sau <LF>, reprezintă începutul unui număr, sfârșitul lui fiind considerat caracterul ce precede <CR>, spațiu sau virgulă. Dacă se întâlnește caracterul ghilimele ("), atunci urmează un șir de caractere încheiat de un alt caracter ghilimele sau de sfârșitul fișierului.

Exemplul 3.20

Să listăm fişierul creat în exemplul 3.19. Programul și rezultatele execuției sînt prezentate în listingul 3.20.

```

5 PRINT "Listarea fisierului secvential CLASA"
10 OPEN "I",#1,"CLASA"
15 IF EOF(1)=-1
    THEN PRINT "Fisierul CLASA este vid"
    ELSE PRINT "Numele si prenumele elevului";:
        PRINT "    romana matematica    fizica";:
        PRINT "    chimie"
20 WHILE EOF(1)<>-1 :
    INPUT #1,ELEV$,MR,MM,MF,MC :
    PRINT USING "\
PRINT USING "    ##    ",MR;MM;MF;MC : "\";ELEV$;:
WEND
25 END
RUN
Listarea fisierului secvential CLASA
Numele si prenumele elevului    romana    matematica    fizica    chimie
Alexandrescu Vasile            9            10            10            9
Florea Doina                    9            8            9            7
Radulescu Luminita            10            9            9            9
Ok

```

Listing 3.20

Remarcă. Pentru alte operații cu fişiere secvențiale a se vedea exemplele din capitolul 5.6.

● Lucrul cu fişiere în acces direct

După cum am văzut, deschiderea fişierelor în acces direct se realizează cu instrucțiunea OPEN cu 'mod=R', ceea ce dă permisiunea atât de scriere cât și de citire a fişierului specificat în OPEN ("numef") de număr logic nrf.

Spre deosebire de modul de lucru cu fişiere secvențiale, în cazul fişierelor cu acces direct, înainte, respectiv după scrierea, respectiv citirea efectivă a unui articol din fişier, trebuie asigurat prin program transferul datelor în și din zona tampon (în / din buffer).

Alocarea de buffere pentru fiecare fişier cu acces direct deschis, se realizează cu instrucțiunea:

```
FIELD [#]nrf[,dim AS varbf1 [,varbf2 ...]]
```

unde: nrf este numărul logic atașat fişierului la deschidere; dim numărul de octeți necesari pentru variabile; varbf₁ variabile de tip șir, semnificînd bufferul (bufferii) alocați fişierului.

Atenție. Numărul total al octeților alocați printr-o instrucțiune FIELD nu trebuie să depășească lungimea înregistrării lg specificate la deschiderea fişierului respectiv.

Observație. Variabilele din lista instrucțiunii FIELD nu pot apare în instrucțiuni LET sau INPUT.

3. Elemente ale limbajului BASIC

Variabilele $varbf_1$ utilizate ca buffer constituie zona de memorie prin care se face schimbul de informații între program și fișierul în acces direct.

Mutarea datelor din memorie într-un buffer de fișier în acces direct se realizează cu instrucțiunile:

```
LSET varbf1 [,varbf2 ...]=exp1 [,exp2 ...]
RSET varbf1 [,varbf2 ...]=exp1 [,exp2 ...]
```

unde: $varbf_1$ reprezintă numele variabilelor declarate cu FIELD; exp_1 expresii cu șiruri de caractere (conțin deci cel mult operatorul '+', concatenare).

O variabilă numerică, pentru a fi depusă în buffer, trebuie convertită în caracter.

Dacă șirul rezultat în urma evaluării expresiilor exp_1 necesită mai puțini octeți decât au fost rezervați cu FIELD pentru lista de variabile $varbf$, atunci LSET îl aliniază la stînga iar RSET la dreapta, restul cîmpurilor fiind completate cu spații.

În afara instrucțiunilor LSET și RSET efectuarea de atribuire asupra variabilelor utilizate ca buffer se mai poate executa cu instrucțiunile PRINT#, PRINT# USING și WRITE#. În cazul instrucțiunii WRITE#, zona buffer neutilizată este umplută cu spațiu după <CR>.

Scrierea efectivă a unui articol într-un fișier în acces direct se realizează cu instrucțiunea:

```
PUT [#]nrf[,nring]
```

unde: nrf este numărul atașat fișierului; $nring$ dacă există, semnifică numărul înregistrării din fișier unde se va face scrierea; dacă lipsește, atunci înregistrarea va primi numărul următor celui dat ultimei înregistrări cu care s-a operat în PUT; $nring$ poate lua valori întregi de la 1 la 32767.

```
DATA „SIR1” , „SIR 2”
```

```
READ A $ , B $
```

```
OPEN „R” , # 1 , „FIS”
```

```
FIELD # 1,10AS A1 $ , 20AS A2 $
```

```
LSET A1 $ = A $
```

```
RSET A2 $ = B $
```

```
PUT = 1 , NR %
```

```
END
```

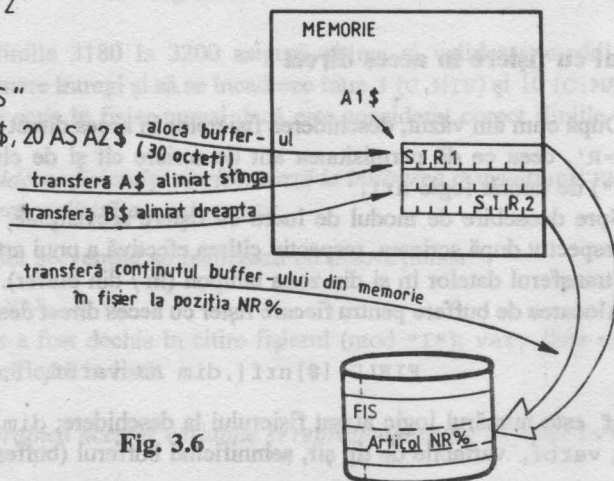


Fig. 3.6

În figura 3.6 este prezentat schematic modul de pregătire și efectuare a scrierii unei înregistrări într-un fișier în acces direct.

Exemplul 3.21

Să se creeze un fișier cu acces direct avînd structura articolelor similară celei din exemplul 3.19.

Indicație. Variabilele numerice utilizate la memorarea mediilor (rotunjite), pentru a putea fi scrise în fişier trebuie convertite în variabile caracter. Utilizăm în acest sens funcția STR\$(x).

Programul este prezentat în listingul 3.21.

```

5 PRINT "Crearea fisierului cu acces direct CLASAD"
10 C.MIN=1 : C.MAX=10
15 OPEN "R",#1,"CLASAD",40
20 FIELD #1,28 AS EL$,3 AS MR$,3 AS MM$,3 AS MF$,3 AS MC$
25 INPUT "Numar inregistrare :",NR%
30 WHILE NR%>0
35 INPUT "Nume :",NUME$
40 INPUT "Prenume :",PREN$
45 ELEV$=NUME$+" "+PREN$
50 C.MAT$="Romana "
55 GOSUB 3180 : MR=C.A
60 C.MAT$="Matematica"
65 GOSUB 3180 : MM=C.A
70 C.MAT$="Fizica "
75 GOSUB 3180 : MF=C.A
80 C.MAT$="Chimie "
85 GOSUB 3180 : MC=C.A
90 PRINT ELEV$;" : ";MR;MM;MF;MC
95 INPUT "Articol corect ? (Y/N) : ",C$
100 IF C$="Y" OR C$="y"
    THEN LSET EL$=ELEV$ :
        RSET MR$=STR$(MR) :
        RSET MM$=STR$(MM) :
        RSET MF$=STR$(MF) :
        RSET MC$=STR$(MC) :
        PUT #1,NR%
105 INPUT "Numar inregistrare :",NR%
110 WEND
115 END
MERGE "C7"
Ok
RUN
Crearea fisierului cu acces direct CLASAD
Numar inregistrare :1
Numele :Alexandrescu
Prenumele :Vasile
Romana ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 10
Fizica ( 1 <= N <= 10 ) = 10
Chimie ( 1 <= N <= 10 ) = 9
Alexandrescu Vasile : 9 10 10 9
Articol corect ? (Y/N) : Y
Numar inregistrare :2
Numele :Florea
Prenumele :Doina
Romana ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 8
Fizica ( 1 <= N <= 10 ) = 9
Chimie ( 1 <= N <= 10 ) = 7
Florea Doina : 9 8 9 7
Articol corect ? (Y/N) : Y
Numar inregistrare :3

```

3. Elemente ale limbajului BASIC

```
Numele      :Radulescu
Prenumele   :Luminita
Romana      ( 1 <= N <= 10 ) = 10
Matematica  ( 1 <= N <= 10 ) = 9
Fizica      ( 1 <= N <= 10 ) = 9
Chimie      ( 1 <= N <= 10 ) = 9
Radulescu Luminita : 10 9 9 9
Articol corect ? (Y/N) : Y
Numar inregistrare :
Ok
```

Listing 3.21

Pentru citirea unei înregistrări dintr-un fișier în acces direct se utilizează, pe lângă instrucțiunea FIELD (necesară pentru alocarea bufferului), instrucțiunea GET cu sintaxa:

```
GET [#]nrf[,nring]
```

unde nrf și nring au aceleași semnificații ca la utilizarea lor în instrucțiunea PUT.

După o instrucțiune GET, pentru citirea caracterelor din buffer și atribuirea valorilor variabilelor corespunzătoare, se pot utiliza instrucțiunile INPUT# și LINE INPUT#.

Exemplul 3.22

Citirea selectivă a articolelor fișierului "CLASAD" creat în exemplul anterior și calculul mediei generale a fiecărui elev.

Programul BASIC și rezultatele execuției sînt prezentate în listingul 3.22

```
10 PRINT "Citire fisier in acces direct"
15 OPEN "R",#1,"CLASAD",40
20 FIELD #1,28 AS EL$,3 AS MR$,3 AS MF$,3 AS MC$
25 INPUT "Numar inregistrare :",NR%
30 WHILE NR%>0
35 GET #1,NR%
40 MED=(VAL(MR$)+VAL(MM$)+VAL(MF$)+VAL(MC$))/4
45 PRINT EL$;" medii:";VAL(MR$);VAL(MM$);VAL(MF$);
50 PRINT VAL(MC$);"=> medie generala :";MED
55 INPUT "Numar inregistrare :",NR%
60 WEND
65 END
```

RUN

Citire fisier in acces direct

Numar inregistrare :1
Alexandrescu Vasile medii: 9 10 10 9 => medie generala : 9.5

Numar inregistrare :3
Radulescu Luminita medii: 10 9 9 9 => medie generala : 9.25

Numar inregistrare :2
Florea Doina medii: 9 8 9 7 => medie generala : 8.25

Numar inregistrare :

Ok

Listing 3.22

3.10 Înlanțuirea programelor

BASIC oferă posibilitatea ca pe parcursul execuției unui program, la un moment dat, să se lanseze în execuție un alt program cu sau fără transferul valorilor tuturor variabilelor sau numai a unora din acestea.

Întreruperea programului curent și predarea controlului altui program se realizează cu comanda CHAIN (prezentată în capitolul 3.2).

Comunicarea între programe se poate realiza în diferite moduri. În continuare ne referim numai transmiterea valorilor variabilelor între programe (apelant-apelat și invers). Aceasta se poate efectua fie utilizând opțiunea ALL a comenzii CHAIN (caz în care se transmit toate variabilele) sau utilizând instrucțiunea COMMON.

Aceasta are sintaxa:

```
COMMON lista
```

unde lista reprezintă lista variabilelor numerice sau de tip șir care se transmit programului apelat. Ele sînt depuse automat într-o arie comună, fiind accesibile tuturor programelor BASIC înlanțuite prin CHAIN.

Remarcă. Variabilele indexate sînt specificate prin identificator urmat de grupul '()'; declararea lor în programul apelat nu mai este necesară.

Atenție. Utilizarea în programul apelant a instrucțiunilor de definire explicită a tipului (DEFINT, DEFNG, DEFDBL sau DEFSTR) are efect local. Pentru asigurarea concordanței tipurilor variabilelor comune celor două programe, în programul apelat este necesară reutilizarea acestora asupra eventualelor variabile implicate.

Exemplul 3.23

Să examinăm un exemplu pur demonstrativ.

În listingul 3.23 se prezintă programul PA în care se definesc două variabile: variabila X, inițializată de la tastatură, și variabila indexată A().

În linia 40 se face înlanțuirea cu programul PA1 (listingul 3.24) din care, la linia 30, se lansează programul PA11 (listingul 3.25).

În programul PA11 se introduc de la tastatură valorile variabilei indexate B() după care, în linia 30 se determină întoarcerea în PA1 (la linia 35). După atribuirile din linia 40, în linia 45 se revine în PA la instrucțiunea cu numărul de linie 45.

O execuție a acestui lanț de programe, pornind de la programul PA, este prezentată în listingul 3.26.

```
10 PRINT "Start program PA"
15 DEFINT A
20 DIM A(2)
25 COMMON A(),X
30 INPUT "X : ",X
35 PRINT "Se lanseaza PA1"
40 CHAIN "PA1"
```

3. Elemente ale limbajului BASIC

```
45 PRINT "Revenire in PA"  
50 PRINT A(0);"^";X;"+";A(1);"^";X-1;"=";A(0)^X+A(1)^(X-1)  
55 END
```

Listing 3.23

```
5 PRINT "Start program PA1" 10 PRINT "Start program PA11"  
10 DEFINT A 20 COMMON B(),X  
15 DIM B(2) 25 FOR I=0 TO 1 :  
20 COMMON A(),B(),X PRINT "B(";I;")"; :  
25 PRINT "Se lanseaza PA11" INPUT " = ",B(I) :  
30 CHAIN "PA11" NEXT  
35 PRINT "Revenire in PA1" 30 CHAIN "PA1",35  
40 FOR I=0 TO 2 : A(I)=B(I) :  
NEXT  
45 CHAIN "PA",45
```

Listing 3.24

```
LOAD "PA"  
RUN  
Start program PA  
X :4  
Se lanseaza PA1  
Start program PA1  
Se lanseaza PA11
```

Listing 3.25

```
Start program PA11  
B ( 0 ) = 4  
B ( 1 ) = 5  
Revenire in PA1  
Revenire in PA  
4 ^ 4 + 5 ^ 3 = 381  
Ok
```

Listing 3.26

3.11 Elemente de grafică

Grafica automată, prin amploarea pe care a căpătat-o în ultimul deceniu, a devenit un domeniu special al informaticii. Evoluția ei, și a mijloacelor tehnice (echipamentelor specializate) este mai mult decât spectaculoasă. Implicațiile sînt multiple reliefîndu-se și în diferitele versiuni și generații ale limbajului BASIC.

Dintre toate categoriile de instrucțiuni BASIC cele grafice cunosc o foarte mare varietate, dar și o puternică dependență de versiunile existente, astfel încît este greu de realizat (și puțin eficient) o tratare globală a lor. Totuși, fiind un domeniu foarte important al informaticii, o prezentare, chiar schematică, a lor este absolut necesară.

Înainte de a prezenta, spre exemplificare, unele facilități grafice disponibile în BASIC-80 (GBASIC) sub CP/M și GW-BASIC sub MS-DOS, se impune precizarea cîtorva noțiuni utilizate curent în aplicații grafice.

● Fereastra, vizor, scalare, decupare

Pentru a înlesni înțelegerea acestor noțiuni considerăm următoarele (figura 3.7):

Fie D un dreptunghi de dimensiune $m \times n$ ($n > 0$ și $m > 0$) spațiul maxim de desen la un moment dat, spațiu în care am realizat (sau intenționăm să realizăm) un desen. Dacă în acest spațiu definim submulțimea F a dreptunghiului $[a, b] \cdot [c, d]$ cu $0 < a, b \leq m$ și $0 < c, d \leq n$ spunem că am definit o fereastră (window) în spațiul D . Atât dimensiunile spațiului D cât și cele ale ferestrei F , sînt exprimate în coordonate care țin de specificul aplicației (de mărimile și unitățile de măsură efectiv utilizate).

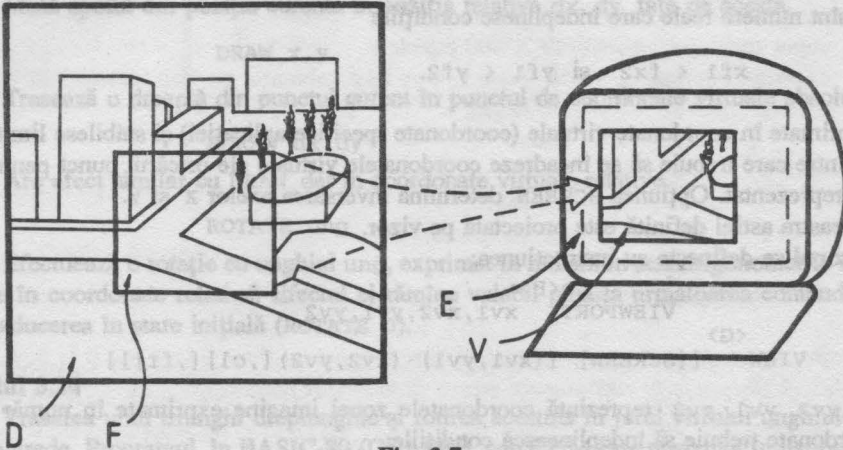


Fig. 3.7

Fie, în același timp, E suprafața totală de $k \times l$ puncte a spațiului util de desen al echipamentului (de exemplu al displayului). O suprafață V dreptunghiulară $[e, f] \cdot [g, h]$ a acestuia (cu $0 < a, b \leq k$ și $0 < g, h \leq l$) se numește vizor (viewport).

Observație. Dimensiunile maxime ale spațiului E (k și l) depind de tipul dispozitivului grafic utilizat.

Atât fereastra F cât și vizorul V pot fi translate oriunde în mulțimile D , respectiv E . De asemenea ele pot fi mărite sau micșorate după necesități, astfel încît pe ecran sau pe o parte a lui să avem la un moment dat întregul desen sau numai un detaliu al acestuia.

Reprezentările grafice presupun suprapunerea originii ferestrei F peste cea a vizorului V . Cum F și V au dimensiuni diferite, rezultă că reprezentarea se va efectua la o anumită scară. Raporturile dintre dimensiunile laturilor ferestrei și cele corespunzătoare ale vizorului se numesc factori de scară. Fie S_x și S_y aceștia. În cazul în care avem de reprezentat suprafețe atunci, dacă $S_y \neq S_x$, imaginea conținută în fereastră va fi reprezentată deformat pe vizor. Pentru evitarea deformărilor în acest caz, se va utiliza un singur factor de scalare:

$$S = \max(S_x, S_y)$$

Încercarea de a desena în afara spațiului ferestrei nu este semnalată ca eroare, dar părțile

3. Elemente ale limbajului BASIC

desenului care depășesc limitele acestea nu se reprezintă. Această operație se numește decupare (clipping).

Definirea ferestrei se efectuează cu instrucțiunea:

```
<B>  
WINDOW xf1,xf2,yf1,yf2  
<G>  
WINDOW [[SCREEN] (xf1,yf1)-(xf2,yf2)]
```

unde $xf1$, $xf2$, $yf1$, $yf2$ sînt limitele abscisei, respectiv ordonatei ferestrei. Coordonatele $xf1$, $xf2$, $yf1$, $yf2$ sînt numere reale care îndeplinesc condițiile

$$xf1 < xf2 \text{ și } yf1 < yf2.$$

Ele sînt exprimate în coordonate virtuale (coordonate specifice aplicației) și stabilesc limitele minime și maxime între care trebuie să se încadreze coordonatele virtuale ale oricărui punct pentru ca acesta să poată fi reprezentat. Opțiunea SCREEN determină inversarea axelor X și Y.

Fereastra astfel definită este proiectată pe vizor.

Vizorul se definește cu instrucțiunea:

```
<B>  
VIEWPORT xv1,xv2,yv1,yv2  
<G>  
VIEW [[SCREEN] [(xv1,yv1)-(xv2,yv2) [,c1] [,fr]]]
```

unde $xv1$, $xv2$, $yv1$, $yv2$ reprezintă coordonatele zonei imagine exprimate în număr de puncte. Aceste coordonate trebuie să îndeplinească condițiile:

$$\begin{aligned} 0 < xv1, xv2 < k \\ 0 < yv1, yv2 < l \end{aligned}$$

unde k și l reprezintă dimensiunile maxime în număr de puncte, ale spațiului de desen al dispozitivului grafic respectiv; cd reprezintă codul culorii (0-3) cu care va fi umplut vizorul, iar fr solicită trasarea frontierei vizorului (cod culoare fr).

Dacă SCREEN este prezent atunci coordonatele grafice se vor calcula relativ la întregul ecran, în caz contrar sînt relative la vizor.

Atît fereastra cît și vizorul pot fi redefinite pe parcursul unui program, însă, la un moment dat există o singură fereastră și un singur vizor, definite prin ultimele instrucțiuni WINDOW și VIEWPORT executate.

Limbajul GW-BASIC permite, de asemenea, definirea de ferestre de text cu instrucțiunea:

```
VIEW PRINT [nrl1 TO nrl2 ]
```

unde nrl_1 și nrl_2 sînt expresii întregi cu valori între 1 și 24 reprezentînd limitele acestuia în număr de linii. După definirea ferestrei de text toate funcțiile de scriere, poziționare, editare, defilare au loc numai în acesta.

 BASIC-80; <G> GW-BASIC

● Instrucțiuni grafice

Principalele instrucțiuni grafice implementate în BASIC-80 (GBASIC) sînt:

MOVE x,y.

Mută spotul (punctul luminos de pe ecran) din poziția în care se află în punctul de coordonate virtuale absolute (x,y).

RMOVE dx,dy

Mută spotul din poziția curentă în poziția relativă dx, dy față de acesta.

DRAW x,y

Trasează o dreaptă din punctul curent în punctul de coordonate virtuale absolute (x,y).

RDRAW dx,dy

Are efect similar cu DRAW dar în coordonate virtuale relative.

ROTATE ung

Efectuează o rotație cu unghiul ung, exprimat în radiani în sens trigonometric direct, a liniilor desenate în coordonate relative. Efectul ei rămîne valabil pînă la următoarea comandă ROTATE sau pînă la aducerea în stare inițială (ROTATE 0).

Exemplul 3.24

Trasarea unui triunghi dreptunghic și rotirea acestuia în jurul vîrfului unghiului drept din 30 în 30 de grade. Programul, în BASIC-80 (GBASIC), sub CP/M este prezentat în listingul 3.27. Liniile 15 la 35 permit citirea și validarea dimensiunilor catetelor (subrutina 3180 a fost prezentată în exemplul 3.19). Variabila RGRD conține valoarea în radiani a unui grad ($180 = \pi$). Liniile 50 și 100 determină comutarea ecranului din mod defilare în mod pagina și invers.

```

5 PRINT "Trasare si rotire triunghi dreptunghic"
10 C.MIN=10 : C.MAX=50 : RGRD=3.1415/180..
15 C.MAT$="Cateta AB"
20 GOSUB 3180 : C=C.A
25 C.MAT$="Cateta AC"
30 GOSUB 3180 : B=C.A
35 PRINT CHR$(27)+"I";CHR$(27)+"2"
40 WINDOW 0,120,0,120
45 VIEWPORT 20,120,0,100
50 MOV 60,60
55 FOR I=0 TO 330 STEP 30
    ROTATE RGRD*I
    GOSUB 80
    NEXT
60 INPUT R$
65 PRINT CHR$(27)+"I";CHR$(27)+"2"
70 END
80 'Trasare triunghi dreptunghic
85 RDRAW C,0
90 RDRAW -C,B
95 RDRAW 0,-B
100 RETURN

```

Listing 3.27

Dintre principalele categorii de instrucțiuni grafice ale interpretorului GW-BASIC amintim:

A. Stabilire caracteristici ecran, cursor, culori

SCREEN [mod] [,ap] [,actp] [,vizp]

Stabilește caracteristicile ecranului de afișat astfel:

mod	expresie numerică cuprinsă între 0 și 255 cu semnificațiile:
0	mod text (25 linii și 80 coloane);
1	rezoluție medie <MR> (320 X 200 puncte);
2	rezoluție înaltă <HR> (640 X 200 puncte);
>3	super rezoluție <SR> (640 X 400 puncte);
ap	expresie întreagă cu valori 0 sau 1, care în combinație cu mod determină:
mod=0	ap=0 => alb-negru;
	ap=1 => color;
mod=1	ap=0 => color
	ap=1 => alb-negru;
mod>1	=> ap se ignoră;
actp	selectare pagină activă;
vizp	selectare pagină virtuală.

Observație. actp și vizp pot avea valori întregi cuprinse între 0 la 7 sau 0 la 3, funcție de lățimea ecranului, și sînt luate în considerare numai în mod text (mod=0).

LOCATE [rnd] [, [col] [,per] [, [lstart] [, lstop]]] (în mod text)

LOCATE [rnd] [, [col] [,per] [,forma]] (în mod grafic)

Poziționează cursorul în rîndul rnd, coloana col și stabilește caracteristicile de vizualizare ale cursorului per (0 - invizibil; 1 - permanent; 2 - 10 clipitor) și dimensiunea acestuia în mod text lstart și lstop (expresii numerice 0-13 pentru cursor utilizator și de scriere forțată și 32-45, modulo 32, pentru cursor utilizator în exclusivitate) sau forma sa forma, în mod grafic (se programează matricea cursorului).

COLOR [cl] [,fc]

COLOR^{<MR>} [fc] [,pl] [,cd] [, [fg] [,ct]]

COLOR^{<HR>} [cd] [,fg] [,ct]

COLOR^{<SR>} [cd] [,fg] [,ct]

Definește culoarea vizorului text fc (expresie numerică 0-31, modulo 15), paleta coloristică pl (expresie numerică 0-255, modulo 2), culoarea desenelor cd (expresie numerică 0-3), culoarea vizorului grafic fg (expresie numerică 0-3) și culoarea caracterelor (a textului) ct (expresie numerică 0-1 în mod text sau 0-7 pentru mod=1 <medie rezoluție>).

Observație. În mod înaltă rezoluție (mod=2 în SCREEN anterior), ct se pune în operație XOR (sau exclusiv) cu harta de alocare ecran. În mod superrezoluție (mod > 3) ct=0 semnifică scriere video normală, ct=1 video invers, iar pentru ct>1 se efectuează XOR cu harta de alocare ecran. În ambele cazuri parametrul fg poate lua numai valorile întregi 0 sau 1.

Codurile culorilor sint:

0 - negru	8 - gri
1 - albastru	9 - albastru aprins
2 - verde	10 - verde aprins
3 - cian	11 - cian aprins
4 - roșu	12 - roșu aprins
5 - magenta	13 - magenta aprins
6 - maron	14 - galben
7 - alb	15 - alb intens

CLS [n]

Umple cu culoarea fondului întregul ecran (n=0), vizorul grafic (n=1) sau vizorul text (n=2).

B. Trasări, hașurări

PSET [STEP] (x,y) [,cl]

PRESET [STEP] (x,y) [,cl]

Provoacă iluminarea punctului de coordonate absolute sau relative (x,y) în culoarea cl (expresie numerică întreagă cu valori 0-3). Dacă parametrul cl lipsește atunci PSET afișează punctul cu culoarea cd declarată în COLOR, iar PRESET cu culoarea fondului (a vizorului grafic <fg în COLOR>) cea ce, de fapt, înseamnă ștergerea lui.

LINE [[STEP] [(x₁, y₁)] - [STEP] (x₂, y₂)] [,cl] [,B[F]] [,stil]]

Trasează o linie între punctele de coordonate relative (dacă STEP este prezent) sau absolute (x₁, y₁) și (x₂, y₂). Opțiunea B determină trasarea unui dreptunghi având colțurile de coordonate (x₁, y₁), (x₂, y₁), (x₂, y₂), (x₁, y₂) cu culoarea cl, iar BF trasarea și umplerea acestuia cu culoarea cl. stil este o valoare întreagă pe 16 biți și reprezintă șablonul de aprindere a punctelor (utilizat pentru linii întrerupte, punctate, etc.).

CIRCLE [STEP] (x,y),r[,cl[,start,stop[,ap]]]

Trasează un (arc de) cerc sau o elipsă de centru (x,y) și rază r (calculate în pixeli), în coordonate absolute sau relative (opțiunea STEP), între unghiurile start și stop (exprimate în radiani (start, stop [-2, 2])), cu culoarea cl și cu raport de aspect asp (implicit 5/6).

Observație. Dacă start sau stop au valori negative se trasează și raza.

DRAW exps

Trasează o imagine grafică descrisă de comenzile GML aparținând expresiei și exps.

Comenzile limbajului GML sînt:

a) stabilire factor de scară (1/<k>):

S<k> unde <k> este o expresie numerică întregă cuprinsă între 1 și 255 (implicit 4);

b) culori:

C<c1> stabilește culoarea c1 (0-3 pentru rezoluție medie, 0-1 rezoluție mare și superrezoluție);

P<c1>, <cf> stabilește culoarea interiorului c1 și a conturului (frontierei) cf ale unei figuri (c1 și cf respectă condițiile comenzii C);

c) deplasări, trasări:

B dacă precede comenzile următoare determină deplasarea fără trasare; dacă nu se efectuează și trasare;

U [<n>] deplasare în sus cu <n>*<k> puncte;

D [<n>] deplasare în jos cu <n>*<k> puncte;

L [<n>] deplasare la stînga cu <n>*<k> puncte;

R [<n>] deplasare la dreapta cu <n>*<k> puncte;

E [<n>] deplasare oblică (SV-NE) cu <n>*<k> puncte;

M<x>, <y> deplasare spot în poziția de coordonate (x,y) absolute sau relative (dacă x și y sînt precedate de '+' sau '-') ; x și y sînt numere întregi;

N deplasare în poziția inițială.

Remarcă. GML oferă de asemenea posibilitatea lansării și execuției unor comenzi specificate într-un subsșir subs cu comanda: X<subs>.

Exemplul 3.25

Trasarea în modul grafic medie rezoluție a unui triunghi isoscel:

```
10 'Trasare triunghi isoscel
15 SCREEN 1
20 DRAW "BM30,30;E50;M90,30;M10,10"
```

Observație. Caracterul ';' utilizat în exemplul de mai sus pentru separarea comenzilor GML este opțional.

```
PAINT [STEP] (x,y) [,tapet[,contur[,fond]]]
```

Determină colorarea sau mozaicarea unei arii grafice începînd din punctul de coordonate absolute sau relative (x,y), cu culoarea tapet (dacă este o expresie numerică întregă cu valori 0-3) sau cu un model (dacă tapet este o expresie șir de 1-64 octeți, formînd un șablon ce reprezintă măști de 8 biți). Coordonatele x, y trebuie să aparțină interiorului figurii de umplut. Conturul acestei figuri se va trasa cu culoarea contur (valoare numerică întregă cu valori 0-3) sau, implicit, cu tapet.

C. Salvări și restaurări de imagini

Salvarea imaginilor se realizează cu instrucțiunea:

```
GET [STEP] (x1, y1) - [STEP] (x2, y2), mat
```

Aceasta transferă imaginea grafică cuprinsă în dreptunghiul cu vîrfurile de coordonate (x₁, y₁), (x₁, y₂), (x₂, y₂), (x₂, y₁) într-un masiv mat de tip întreg care trebuie să aibă dimensiunea minimă (în octeți):

$$4 + \text{INT}((dx * bpp + 7) / 8) * dy$$

unde bpp reprezintă numărul de biți pe pixel (1 pentru rezoluție mare și super rezoluție <mod=2, respectiv mod=3 în SCREEN>, respectiv 2 pentru rezoluție medie <'mod'=1>), iar dx și dy reprezintă dimensiunile în puncte (pixeli) pe orizontală și verticală ale zonei imagine de salvat.

Restaurarea imaginilor grafice salvate se realizează cu instrucțiunea:

```
PUT (x,y), mat[, verb]
```

Aceasta transferă pe ecran, într-un dreptunghi avînd coordonatele colțului de NV (stînga sus), perechea (x,y), imaginea memorată în tabloul numeric mat interacționîndu-se cu imaginea deja existentă funcție de parametrul verb, ce poate lua valorile:

- | | |
|--------|--------------------------------------------------------------------------------------------|
| PSET | transferă punct cu punct, cu respectarea culorii din momentul salvării imaginii; |
| PRESET | transferă punct cu punct reversul imaginii salvate; |
| AND | efectuează operația logică SI între elementul de imagine existent și cel salvat în mat; |
| OR | suprapune imaginea salvată peste cea existentă; |
| XOR | efectuează operația logică SAU EXCLUSIV între elementul de imagine existent și cel salvat. |

4. SPRE O PROGRAMARE MODULARĂ ȘI STRUCTURATĂ

După cum am văzut în capitolul 2 orice algoritm logico-matematic poate fi astfel întocmit și reprezentat încât să conțină numai trei tipuri de structuri de control fundamentale, bine definite: liniare, alternative și repetitive, cu respectarea principalei caracteristici (reguli), de a avea o singură intrare și o singură ieșire.

Din multiplele facilități oferite de utilizarea acestor structuri în descrierea algoritmilor, o bună lizibilitate, reducerea considerabilă a posibilității de a comite erori, ușurința în urmărire, verificare și, eventual, corectare, o eficiență sporită etc., subliniem, în special, simplitatea cuplării și imbricării acestora în descrierea algoritmilor complecși. Datorită proprietății fundamentale (o singură intrare, o singură ieșire) se creează posibilitatea ca, indiferent de complexitatea unui algoritm, în ultimă instanță, acesta să poată fi reprezentat prin câteva structuri simple, conținând unul sau mai multe module.

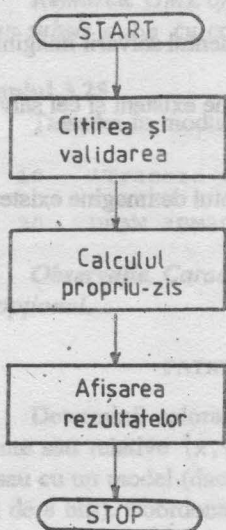


Fig. 4.1

În exemplele prezentate în capitolul 2 gruparea acțiunilor în module, deși nu s-a subliniat în mod special, nu s-a efectuat la întâmplare ci pornind de la funcțiile pe care le îndeplinea grupul respectiv de acțiuni (operații) în cadrul algoritmului. Astfel în capitolul 2.4 algoritmul pentru calculul valorii funcției $f(x)$, (figura 2.15) cuprinde trei module: un modul de intrare (citirea valorii variabilei x), un modul de ieșire (afișarea soluției) și modulul de calcul propriu-zis, module care, prin funcțiile lor, descriu în mod sintetic însăși etapele și modul de rezolvare al problemei analizate. În mod similar stau lucrurile și în cazul exemplului 2.5 (figura 2.19).

Șigur, acestea sînt probleme foarte simple care pot fi rezolvate și direct. Ne-am oprit însă asupra lor tocmai pentru a sublinia cea mai sintetică modalitate de descriere a unei probleme, care ar trebui să fie și prima etapă în soluționarea ei, și anume stabilirea funcțiilor generale și a modulelor care să le îndeplinească. Acestea ar fi (figura 4.1):

- funcția de citire a valorilor variabilelor de intrare, efectuată de modulul conținînd blocul (blocurile) de intrare și eventualele validări;
- funcția de calcul propriu-zis, efectuată de modulul (modulele) de calcul;
- funcția de afișare a soluțiilor, efectuată de modulul (modulele) conținînd

blocurile de ieșire.

După cum vom vedea și în continuare acestea sînt module universal valabile în descrierea oricărui algoritm. Diferențieri apar abia în următoarele etape ale rezolvării problemei, etape în care,

pornind de la funcția generală a fiecărui modul se trece la explicitarea într-un număr mai mare sau mai mic de 'submodule' interconectate conform specificului problemei de soluționat.

În final, prin explicitarea și interconectarea modulelor se obține algoritmul (descriș utilizând schemele logice, în pseudocod sau direct într-un limbaj de programare) soluționării problemei.

Cum în explicitarea modulelor s-a pornit de la general la particular, de la funcția (funcțiile) ce urmează a fi îndeplinite și prin faptul că modulele păstrează importanța proprietate de a avea o singură intrare și o singură ieșire, funcție de tehnica și experiența programatorului, ele capătă un grad mai mic sau mai mare de generalitate, astfel încât pot fi utilizate ca atare și în alte puncte ale algoritmului (programului), precum și în soluționarea altor probleme, ceea ce constituie un avantaj deosebit. O astfel de situație a fost întâlnită în exemplul 2.7 (capitolul 2), când, în descrierea algoritmului lui Euclid s-a utilizat procedura SREST realizată în exemplul 2.6, fapt evidențiat în figurile 2.20 și 2.21b.

Tehnica de realizare a programelor modulare poartă numele de programare modulară și urmărește reducerea complexității realizării și testării programelor, prin descompunerea acestora în module interconectate în conformitate cu o ierarhie bine precizată, determinată de problema soluționată.

Modularizarea (realizarea de module program) se poate obține și fără utilizarea structurilor fundamentale prezentate în capitolul 2 dar, în acest caz, performanțele sînt mult mai modeste. Din această cauză optăm ca realizarea modulelor să se efectueze numai prin utilizarea structurilor de control fundamentale prezentate, conform principiilor programării structurate.

Programarea structurată este definită [13] *ca activitate ce urmărește elaborarea disciplinată a unui program (modul de program) pe baza unor structuri de control fundamentale și date reprezentative problemei rezolvate, în scopul obținerii unui produs program care reflectă clar algoritmul și datele corespunzătoare problemei, este inteligibil, adaptabil și testabil.*

Un prim pas spre realizarea de programe structurate și modularizate a fost făcut pe parcursul capitolului 2, prin însăși modul de abordare și prezentare al modalităților de descriere ale algoritmilor prin utilizarea schemelor logice sau în pseudocod.

Din cele prezentate în capitolul 3 observăm însă că limbajul BASIC nu oferă facilități deosebite pentru realizarea unor astfel de programe. Totuși, cu toate restricțiile impuse, realizarea unor programe structurate și modularizate este posibilă.

Să examinăm câteva exemple.

Exemplul 4.1

În capitolul 2, exemplul 2.7, a fost prezentat algoritmul lui Euclid pentru determinarea c.m.m.d.c. al două numere naturale diferite de zero. De data aceasta să determinăm c.m.m.d.c. fără a mai efectua împărțirea cu rest, ci prin scăderi succesive. Astfel, pentru a determina c.m.m.d.c. al numerelor N_1 și N_2 , acestea se compară între ele și, dacă sînt egale, atunci c.m.m.d.c. este unul dintre ele, dacă nu, se scade cel mai mic din cel mai mare și se reeefectuează comparația; dacă au devenit egale, atunci acesta este c.m.m.d.c. al lor; dacă nu, se efectuează o nouă scădere, și așa mai departe. Dacă, în final, se obține identitatea $1=1$ atunci numerele sînt prime între ele.

O variantă de program pentru determinarea în acest mod a c.m.m.d.c. al numerelor N_1 și N_2 este prezentată în listingul 4.1 avînd schema logică în figura 4.2.

4. Spre o programare modulară și structurată

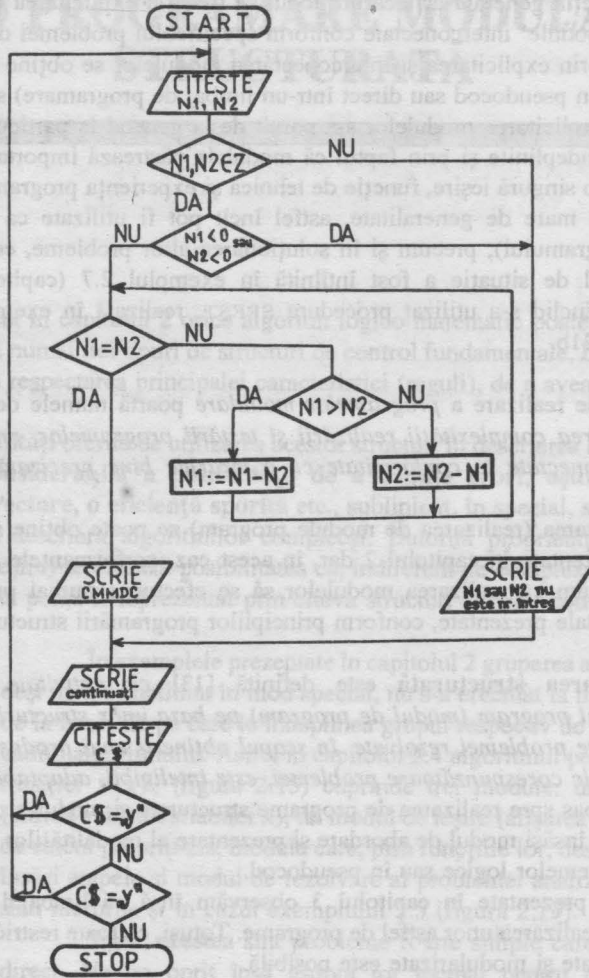


Fig. 4.2

```

10 PRINT "Programul determina C.M.M.D.C al N1 si N2"
20 INPUT "N1 = ", N1
30 INPUT "N2 = ", N2
40 IF INT(N1) <> N1 OR INT(N2) <> N2 THEN GOTO 110
50 IF N1 <= 0 OR N2 <= 0 THEN GOTO 110
60 IF N1 = N2 THEN GOTO 100
70 IF N1 > N2 THEN GOTO 90
80 N2=N2-N1 : GOTO 60
90 N1=N1-N2 : GOTO 60
100 PRINT "C.M.M.D.C. = "; N1 : GOTO 120
110 PRINT "N1 sau N2 nu este numar natural pozitiv"
120 INPUT "Continuati ? (Y/N) : ", C$
130 IF C$ = "Y" THEN GOTO 20
  
```



```

140. IF CŞ = "y" THEN GOTO 20
150. STOP
    
```

Listing 4.2

Deși simplu, în această variantă de redactare, programul este oarecum dificil de urmărit.

Să rezolvăm acum aceeași problemă utilizând tehnicile programării structurate și modulare.

Prin program ne propunem să rezolvăm următoarele funcții:

- citirea și validarea valorilor variabilelor de intrare;
- determinarea c.m.m.d.c.;
- afișarea c.m.m.d.c.

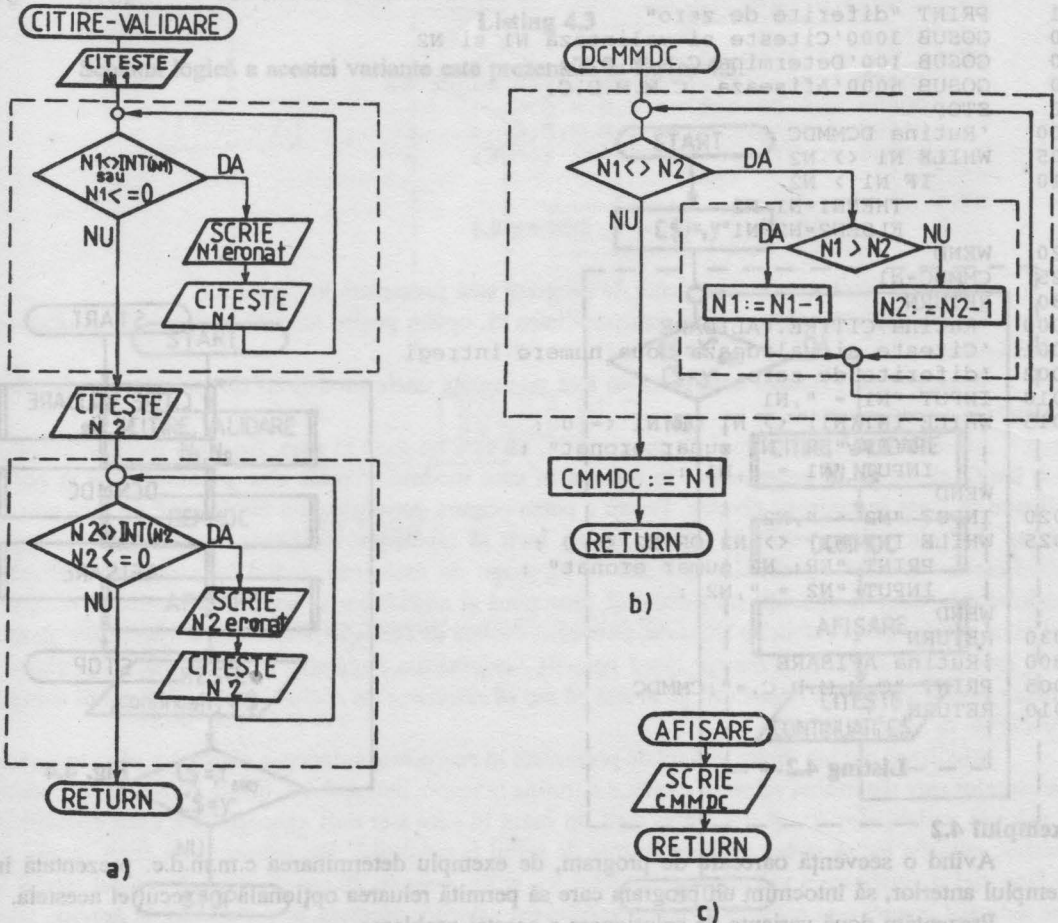


Fig. 4.3

4. Spre o programare modulară și structurată

Pentru realizarea acestor funcții vom utiliza trei module de program:

- modulul CITIRE.VALIDARE (figura 4.3a) pentru citirea și validarea valorilor variabilelor N_1 și N_2 ; modulul conține două structuri repetitive WHILE-DO, care determină repetarea introducerii valorilor fiecărei variabile atât timp cât acestea nu aparțin mulțimii valorilor admise (numerele naturale diferite de zero);

- modulul DCMMDC (figura 4.3b) pentru determinarea c.m.m.d.c. al numerelor N_1 și N_2 ; modulul este alcătuit dintr-o structură WHILE-DO care conține la rândul ei o structură IF-THEN-ELSE;

- modulul AFISARE (figura 4.3c) pentru afișarea soluției; conține instrucțiunea PRINT.

Cu aceste elemente schema logică devine cea din figura 4.4. Programul BASIC corespunzător (incluzând și modulele aferente) este prezentat în listingul 4.2.

```
10 PRINT "Determinarea C.M.M.D.C. al doua numere naturale"
11 PRINT "diferite de zero"
20 GOSUB 3000'Citeste si valideaza N1 si N2
30 GOSUB 100'Determina C.M.M.D.C.
40 GOSUB 5000'Afiseaza C.M.M.D.C.
50 STOP
100 'Rutina DCMMDC
105 WHILE N1 <> N2
110 IF N1 > N2
    THEN N1=N1-N2
    ELSE N2=N2-N1
120 WEND
125 CMMDC=N1
130 RETURN
3000 'Rutina CITIRE.VALIDARE
3001 'Citeste si valideaza doua numere intregi
3002 'diferite de zero
3010 INPUT "N1 = ",N1
3015 WHILE INT(N1) <> N1 OR N1 <= 0 :
    PRINT "ER: N1 numar eronat" :
    INPUT "N1 = ",N1 :
WEND
3020 INPUT "N2 = ",N2
3025 WHILE INT(N2) <> N2 OR N2 <= 0 :
    PRINT "ER: N2 numar eronat" :
    INPUT "N2 = ",N2 :
WEND
3030 RETURN
5000 'Rutina AFISARE
5005 PRINT "C.M.M.D.C.=";CMMDC
5010 RETURN
```

Listing 4.2

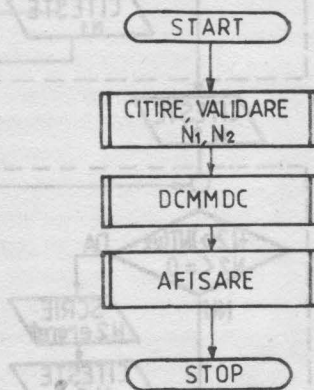


Fig. 4.4

Exemplul 4.2

Având o secvență oarecare de program, de exemplu determinarea c.m.m.d.c. prezentată în exemplul anterior, să întocmim un program care să permită reluarea opțională a execuției acesteia.

Prezentăm două variante de soluționare a acestei probleme.

O primă variantă este cea ilustrată prin programul din listingul 4.3 și constă în inserarea după secvența de program propriu-zisă (liniile 20-40) a două instrucțiuni: o instrucțiune INPUT (linia 50)

prin care se solicită și se primește un răspuns privind opțiunea de continuare, și o instrucțiune IF-GOTO (linia 60) prin care, în urma evaluării expresiei logice, determină, fie salt la linia 20 și reluarea secvenței pentru noi valori ale lui N1 și N2, fie trecera la linia 70 (STOP).

```

1 'Varianta de executie repetata
2 'a unei secvente de program
10 PRINT "Determinarea C.M.M.D.C. al doua numere naturale"
11 PRINT "diferite de zero"
20 GOSUB 3000'Citeste si valideaza N1 si N2
30 GOSUB 100'Determina C.M.M.D.C.
40 GOSUB 5000' Afiseaza C.M.M.D.C.
50 INPUT "Continuati ? (Y/N) : ",C$
60 IF C$ = "Y" OR C$ = "y" GOTO 20
70 STOP
    
```

Listing 4.3

Schema logică a acestei variante este prezentată în figura 4.5.

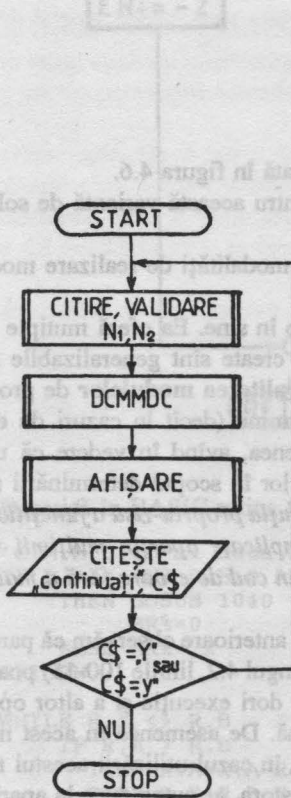


Fig. 4.5

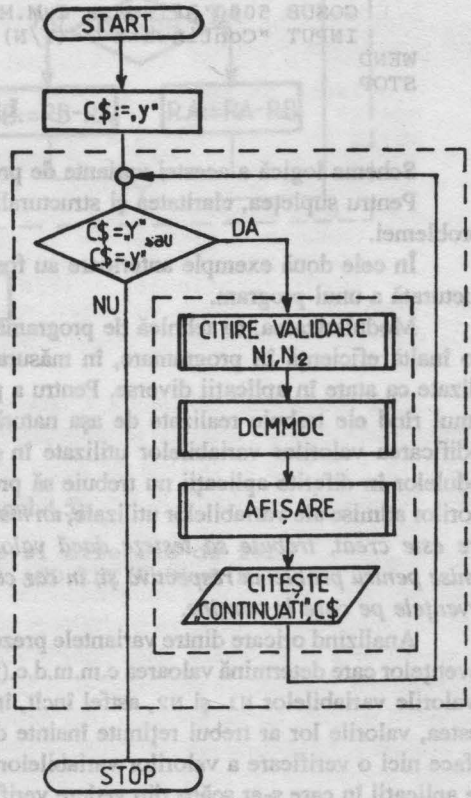


Fig. 4.6

4. Spre o programare modulară și structurată

O altă variantă de soluționare a problemei este includerea secvenței de executat într-o structură de tip WHILE-DO (listingul 4.4). În acest caz, deoarece structura WHILE-DO presupune întâi evaluarea expresiei logice (condiției) și apoi, dacă aceasta este adevărată, parcurgerea secvenței, instrucțiunea WHILE (linia 30) este necesar a fi precedată de o instrucțiune prin care să se pregătească evaluarea în sensul dorit a expresiei logice respective (în exemplul prezentat în listingul 4.4 acesta se realizează prin instrucțiunea de atribuire din linia 20). După intrarea în ciclul pregătirea evaluării expresiei logice a instrucțiunii WHILE se efectuează în linia 70 prin instrucțiunea INPUT. Funcție de valoarea primită de variabila C\$, secvența se poate relua sau nu.

```
1 'Varianta structurata de executie repetata
2 'a unei secvente de program
10 PRINT "Determinarea C.M.M.D.C. al doua numere naturale"
11 PRINT "diferite de zero"
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40 GOSUB 3000'Citire si validare N1 si N2
50 GOSUB 100'Determina C.M.M.D.C.
60 GOSUB 5000'Afiseaza C.M.M.D.C.
70 INPUT "Continuati ? (Y/N) : ",C$
80 WEND
90 STOP
```

Listing 4.4

Schema logică a acestei variante de program este prezentată în figura 4.6.

Pentru suplețea, claritatea și structuralitatea ei, optăm pentru această variantă de soluționare a problemei.

În cele două exemple anterioare au fost prezentate unele modalități de realizare modulară și structurată a unui program.

Modularizarea, ca tehnică de programare, nu este un scop în sine. Ea oferă multiple avantaje și o înaltă eficiență în programare, în măsura în care modulele create sînt generalizabile și pot fi utilizate ca atare în aplicații diverse. Pentru a putea asigura generalitatea modulelor de program, în primul rînd ele trebuie realizate de așa natură, încît să nu determine (decît în cazuri de excepție) modificarea valorilor variabilelor utilizate în program; de asemenea, avînd în vedere că utilizarea modulelor în diferite aplicații nu trebuie să presupună și analiza lor în scopul determinării mulțimii valorilor admise ale variabilelor utilizate, *un modul, înainte de execuția propriu-zisă a funcțiilor pentru care este creat, trebuie să testeze dacă valorile variabilelor implicate aparțin mulțimii valorilor admise pentru problema respectivă și, în caz că nu, să returneze un cod de eroare fără a mai executa secvențele pe care le conține.*

Analizînd oricare dintre variantele prezentate în exemplele anterioare observăm că parcurgerea secvențelor care determină valoarea c.m.m.d.c. (rutina DCMMDC, listingul 4.2, liniile 100-15) poate afecta și valorile variabilelor N1 și N2, astfel încît, în cazul în care s-ar dori execuția și a altor operații cu acestea, valorile lor ar trebui reținute înainte de saltul la subrutină. De asemenea în acest modul nu se face nici o verificare a valorilor variabilelor N1 și N2, ceea ce, în cazul utilizării acestui modul în alte aplicații în care s-ar scăpa din vedere verificarea valorilor acestora, ar putea duce la apariția unor erori.

Cum am putea realiza un modul (rutină) cu un grad cît mai mare de generalitate ?

Exemplul 4.3

Să se generalizeze rutina de determinare a c.m.m.d.c. utilizată în exemplele anterioare.

Pentru aceasta vom utiliza variabilele $R.A$, $R.B$ și $R.C$ ca variabile interne pentru determinarea c.m.m.d.c., precum și variabila $ER\%$ pentru transmiterea codului de eroare la întoarcerea din rutină. Dacă execuția rutinei a decurs normal $ER\%$ va conține valoarea 0, în caz contrar conținând valoarea -2.

Schema logică a rutinei DCMMDC devine (figura 4.7):

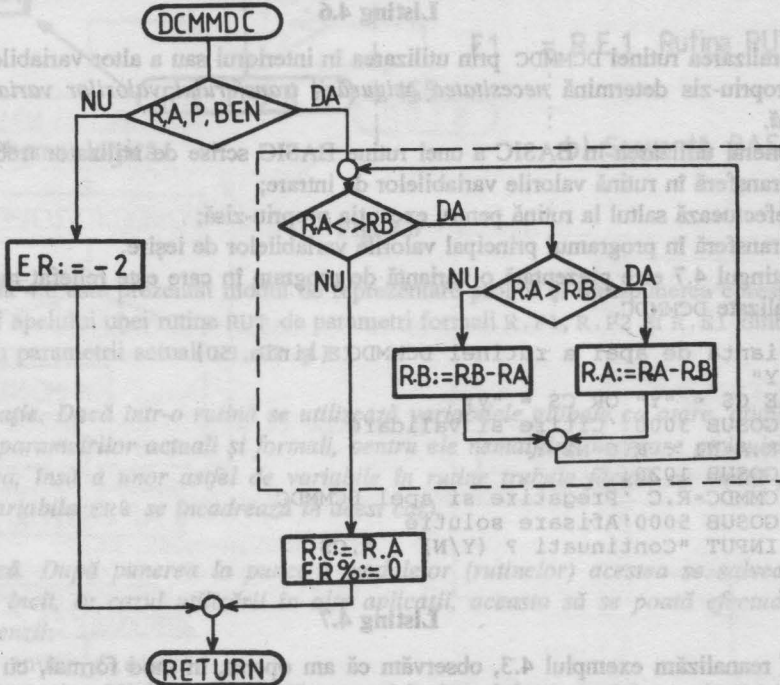


Fig. 4.7

Descrișă în BASIC rutina DCMMDC este (listingul 4.5):

```

1020 'C.M.M.D.C. al doua numere naturale si diferite de zero
1025 IF R.A>0 AND R.B>0 AND INT(R.A)=R.A AND INT(R.B)=R.B
    THEN GOSUB 1040 :
    ER%=0
    ELSE ER%=-2
1030 RETURN
1035 '
1040 WHILE R.A <> R.B
1045 IF R.A > R.B
    THEN R.A=R.A-R.B
    ELSE R.B=R.B-R.A
1050 WEND
1055 R.C=R.A
1060 RETURN
  
```

Listing 4.5

4. Spre o programare modulară și structurată

Utilizarea unei variabile de eroare impune refacerea și a rutinei de afișare, care acum devine (listingul 4.6):

```
5000 'Rutina AFISARE
5005 IF ER% = 0
      THEN PRINT "C.M.M.D.C. = "; CMMDC
      ELSE PRINT "ER = "; ER%
5010 RETURN
```

Listing 4.6

Generalizarea rutinei DCMMDC prin utilizarea în interiorul sau a altor variabile decât cele din programul propriu-zis determină *necesitatea asigurării transferului valorilor variabilelor spre și dinspre rutină.*

În general utilizarea în BASIC a unei rutine BASIC scrise de utilizator trebuie să decurgă astfel:

- se transferă în rutină valorile variabilelor de intrare;
- se efectuează saltul la rutină pentru execuția propriu-zisă;
- se transferă în programul principal valorile variabilelor de ieșire.

În listingul 4.7 este prezentată o variantă de program în care este reliefat modul de apel al rutinei generalizate DCMMDC.

```
10 'Varianta de apel a rutinei DCMMDC (linia 50)
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40   GOSUB 3000 'Citire si validare
50   R.A=N1 : R.B=N2 :
      GOSUB 1020:
      CMMDC=R.C 'Pregatire si apel DCMMDC
60   GOSUB 5000'Afisare solutie
70   INPUT "Continuati ? (Y/N) : ",C$
80 WEND
90 STOP
```

Listing 4.7

Dacă reanalizăm exemplul 4.3, observăm că am operat, în mod formal, cu două tipuri de variabile: variabilele generale ale programului (N1, N2, CMMDC și ER%) și variabilele specifice rutinei DCMMDC (R.A, R.B și R.C). Deși din punct de vedere al majorității variantelor de BASIC nu există această departajare (de regulă, în BASIC toate variabilele sînt globale), spre a înlesni însușirea și a altor limbaje de programare și pentru uniformitate, vom numi variabilele programului propriu-zis **variabile globale**, iar cele utilizate numai în rutine **variabile locale**. De asemenea, din același considerent și, în plus, pentru a ușura descrierea algoritmilor utilizînd schemele logice sau pseudocodul, descrierea apelurilor la rutine va fi efectuată utilizînd blocul de procedură prezentat în capitolul 2.2, prin specificarea în paranteză, după numele rutinei, a variabilelor implicate, într-o listă numită **lista de parametrii**. Variabilele programului principal care apar în lista parametrilor de apel se numesc **parametrii actuali**, iar cei specifici (din rutină) corespunzători lor, se numesc **parametrii formali**. Între ordinea și tipul parametrilor actuali și formali trebuie să fie o corespondență biunivocă.

Observație importantă. Cum în majoritatea variantelor de BASIC nu există nici o distincție între variabilele globale și locale, transferul de valori între parametrii actuali și formali utilizați nu se efectuează automat și, ca atare, în program trebuie să apară efectiv operațiile de atribuire respective.

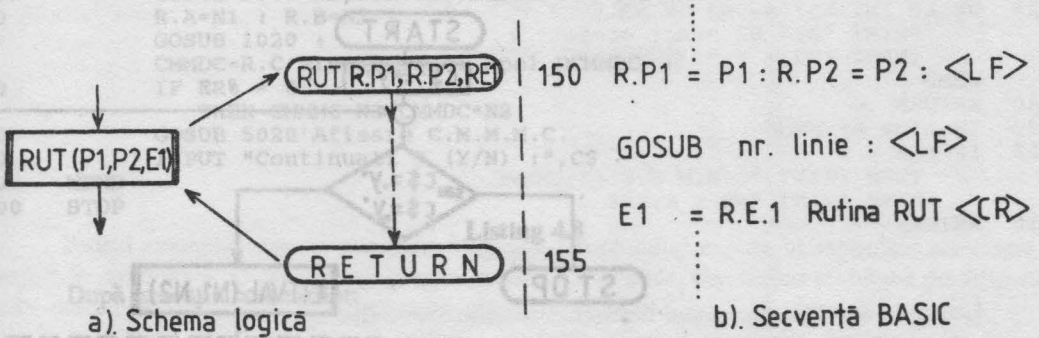


Fig. 4.8

În figura 4.8 este prezentat modul de reprezentare propus și transpunerea corespunzătoare în BASIC în cazul apelului unei rutine RUT de parametri formali R.P1, R.P2 și R.E1 dintr-un program oarecare, pentru parametrii actuali P1, P2 și E1.

Observație. Dacă într-o rutină se utilizează variabilele globale ca atare, atunci ele nu mai apar în listele parametrilor actuali și formali, pentru ele nemaifiind necesare atribuiri (nu ar avea sens). Utilizarea, însă a unor astfel de variabile în rutine trebuie făcută cu multă precauție (În exemplul 4.3 variabila ER% se încadrează în acest caz).

Remarcă. După punerea la punct a modulelor (rutinelor) acestea se salvează în fișiere separate astfel încât, în cazul utilizării în alte aplicații, aceasta să se poată efectua prin simpla execuție a comenzii:

```
MERGE "nume_fisier".
```

Pentru a mai sublinia încă o dată avantajele lucrului cu module (rutine) mai prezentăm următorul exemplu.

Exemplul 4.4

Să se întocmească un program pentru determinarea c.m.m.m.c al două numere naturale diferite de zero.

Programul trebuie să îndeplinească următoarele funcții:

- citirea și validarea a două numere naturale diferite de zero;
- determinarea c.m.m.m.c.;
- afișarea rezultatului;
- reluarea opțională a secvenței de program pentru alte valori ale variabilelor de intrare.

Observăm că prima funcție o avem deja programată în rutina CITIRE.VALIDARE (listîngul 4.2, liniile 3000-3030), rutina pe care o presupunem salvată în fișierul CITVAL.

Schema logică de determinare a c.m.m.m.c al două numere naturale diferite de zero este cea din figura 4.9. Avînd deja rutina DCMMDC pentru determinarea c.m.m.d.c. realizarea acestei funcții este extrem de simplă.

4. Spre o programare modulară și structurată

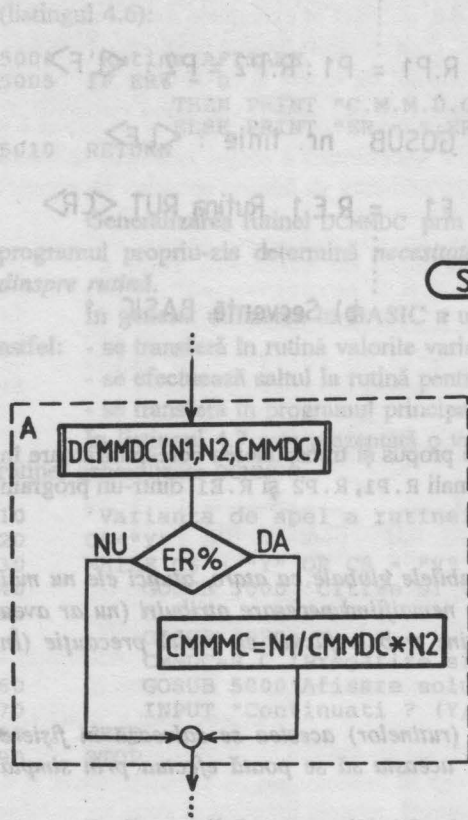


Fig. 4.9

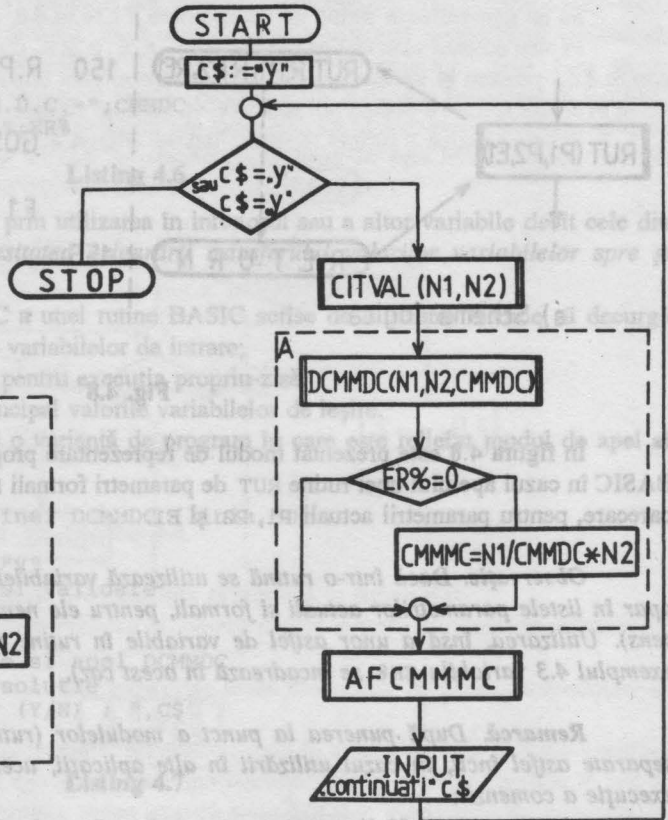


Fig. 4.10

Pentru afișarea soluției folosim o rutină asemănătoare celei prezentate în exemplul 4.3.

În vederea asigurării posibilității de reluare opțională a execuției secvenței dorite vom utiliza structura WHILE-DO descrisă în exemplul 4.2.

Schema logică a programului este cea din figura 4.10. În blocul A (încadrat în figura 4.10 cu linie întreruptă) regăsim algoritmul de determinare a c.m.m.m.c. din figura 4.9. Practic, cu o foarte mică excepție (o modificare nesemnificativă în rutina de afișare), aceasta este singura parte nouă față de cele prezentate în exemplul 4.3, necesară pentru soluționarea problemei propuse în acest exemplu.

Programul propriu-zis este cel prezentat în listingul 4.8.

```

10 PRINT "Determinarea C.M.M.M.C. al doua numere"
11 PRINT "diferite de zero "
20 C$="Y"
  
```



```

30  WHILE CŞ = "Y" OR CŞ = "y"
40  GOSUB 3000'Citire si validare - CITVAL -
50  R.A=N1 : R.B=N2 :
    GOSUB 1020 :
    CMMDC=R.C'Pregatire si apel DCMMDC
60  IF ER% = 0
    THEN CMMC=N1/CMMDC*N2
70  GOSUB 5020'Afisare C.M.M.M.C.
80  INPUT "Continuati ? (Y/N) :","CŞ
90  WEND
100 STOP

```

Listing 4.8

După execuția comenzilor:

```

MERGE "CITVAL"
MERGE "DCMMDC"
MERGE "AFCMMC"

```

acesta devine (listigul 4.9):

```

10  PRINT "Determinarea C.M.M.M.C. al doua numere naturale"
11  PRINT "diferite de zero "
20  CŞ="Y"
30  WHILE CŞ = "Y" OR CŞ = "y"
40  GOSUB 3000'Citire si validare - CITVAL -
50  R.A=N1 : R.B=N2 :
    GOSUB 1020 :
    CMMDC=R.C'Pregatire si apel DCMMDC
60  IF ER% = 0
    THEN CMMC=N1/CMMDC*N2
70  GOSUB 5020'Afisare C.M.M.M.C.
80  INPUT "Continuati ? (Y/N) :","CŞ
90  WEND
100 STOP
1020 'C.M.M.D.C. al doua numere naturale si diferite de zero
1025 IF R.A>0 AND R.B>0 AND INT(R.A)=R.A AND INT(R.B)=R.B
    THEN GOSUB 1040 :
    ER%=0
    ELSE ER%=-2
1030 RETURN
1035 '
1040 WHILE R.A <> R.B
1045 IF R.A > R.B
    THEN R.A=R.A-R.B
    ELSE R.B=R.B-R.A
1050 WEND
1055 R.C=R.A
1060 RETURN
3000 'Rutina CITIRE.VALIDARE
3001 'Citeste si valideaza doua numere intregi
3002 'diferite de zero
3010 INPUT "N1 = ",N1
3015 WHILE INT(N1) <> N1 OR N1 <= 0 :
    PRINT "ER: N1 nunar eronat" :
    INPUT "N1 = ",N1 :

```

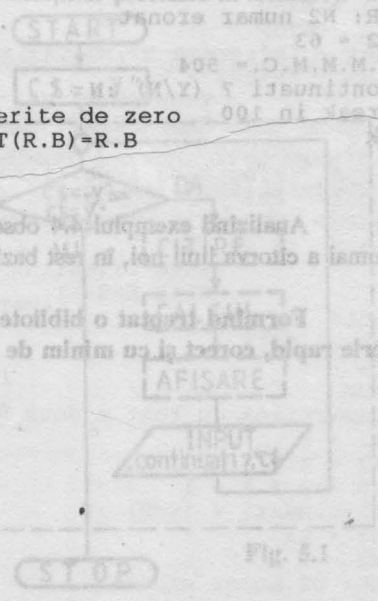


Fig. 5.1

4. Spre o programare modulară și structurată

```
WEND
3020 INPUT "N2 = ",N2
3025 WHILE INT(N2) <> N2 OR N2 <= 0 :
    PRINT "ER: N2 numar eronat" :
    INPUT "N2 = ",N2 :
WEND
3030 RETURN
5020 'Rutina AFCMMMM
5025 IF ER% = 0
    THEN PRINT "C.M.M.M.C.=";CMMM
    ELSE PRINT "ER = ";ER%
5030 RETURN
```

Listing 4.9

Lansat în execuție programul furnizează soluții corecte (listingul 4.10).

```
RUN
Determinarea C.M.M.M.C. al doua numere naturale
diferite de zero
N1 = 4
N2 = 6
C.M.M.M.C.= 12
Continuati ? (Y/N) :Y
N1 = -1
ER: N1 numar eronat
N1 = 1
N2 = 8
C.M.M.M.C.= 8
Continuati ? (Y/N) :Y
N1 = 24
N2 = 0
ER: N2 numar eronat
N2 = 63
C.M.M.M.C.= 504
Continuati ? (Y/N) :N
Break in 100
Ok
```

Listing 4.10

Analizând exemplul 4.4 observăm că pentru soluționarea problemei a fost necesară scrierea numai a câtorva linii noi, în rest bazându-ne pe utilizarea unor module create anterior.

Formind treptat o bibliotecă de astfel de module se poate ajunge la performanța de a scrie rapid, corect și cu minim de efort programe din ce în ce mai complexe.

5. APLICAȚII

Pentru exemplificarea și mai buna înțelegere a noțiunilor expuse în capitolele anterioare se prezintă în continuare 40 de aplicații grupate în șase subcapitole, cuprinzând probleme de dificultate medie desprinse din tematica învățămîntului gimnazial și liceal.

Aplicațiile sînt concepute și realizate modular utilizînd structurile fundamentale cu care operează programarea structurată, astfel încît, pe lîngă liniile program specifice fiecăreia (cu etichete de linii între 5 și 999), au fost definite și utilizate module generale de calcul (cu etichete de linii între 1000 și 2999), de citire și validare a unor date (cu etichete de linii între 3000 și 4999) și de afișare a rezultatelor (avînd etichete de linii în intervalul 5000 la 5999).

Necesitatea unui cît mai eficient control al modului de desfășurare al diferitelor rutine utilizate în construcțiile aplicațiilor prezentate în acest capitol a impus definirea variabilei ER $\&$ care conține în orice moment codul ultimei erori depistate. Definirea modulelor s-a efectuat o singură dată, în aplicațiile care le utilizează prima dată. Funcțiile și principalele caracteristici ale modulelor utilizate (denumire simbolică, parametri de intrare și / sau ieșire, variabilele "globale" utilizate, codurile de eroare returnate etc.) sînt prezentate în anexa A.

Mulțimile valorilor etichetelor liniilor modulelor fiind disjuncte, ele se pot utiliza în programe și în alte combinații decît cele prezentate (cu respectarea însă a condițiilor precizate în anexa 1).

Structurile generale (programele pîncipale) ale aplicațiilor 1 - 38 respectă, cu unele particularități, structura de program prezentată în figura 5.1.

După cum se poate observa cu ușurință este vorba de o structură WHILE-DO, structură prezentată pe larg în 3.8.1. S-a ales acest mod de abordare deoarece oferă utilizatorului posibilitatea de a rămîne în conversație cu calculatorul atît timp cît dorește; pot fi studiate astfel diferite aspecte generale sau particulare ale problemelor modelate fără a mai fi necesară relansarea programelor pentru fiecare set de date particular.

Pentru separarea, pe cît posibil, a variabilelor specifice modulelor de cele cu care operează programele propriu-zise, identificatorii acestora (considerați din punct de vedere logic 'locali') sînt precedați de R., C. și respectiv S.. Aceasta impune, de regulă, ca la utilizare să se efectueze explicit transferul valorilor parametrilor implicați, înainte și / sau după apelarea modulelor. Deși relativ incomodă, această modalitate de realizare a modulelor le conferă acestora un grad relativ mare de

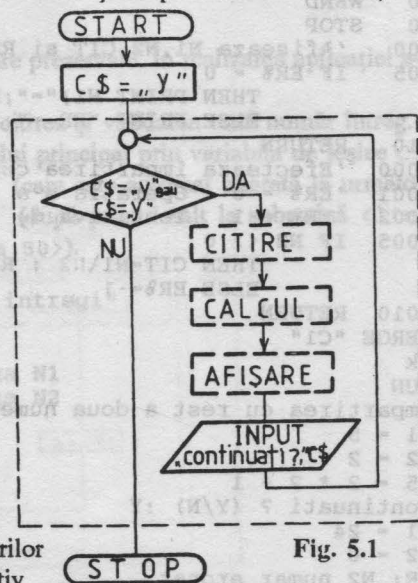


Fig. 5.1

5. Aplicații

generalitate și o oarecare independență față de programele apelante.

Reunirea modulelor într-o bibliotecă și dezvoltarea treptată a sa, se poate dovedi deosebit de eficientă pentru utilizatorul BASIC, înlăturând parțial inconvenientele datorate lipsei (în majoritatea covârșitoare a generațiilor și versiunilor BASIC) procedurilor înlătinate în alte limbaje de programare evaluate.

Pentru a conferi un cât mai profund caracter practic și a înlesni parcurgerea, înțelegerea și analiza aplicațiilor prezentate, acestea sînt însoțite de rezultatele execuțiilor efectuate cu date de test.

5.1 Numere întregi, baze de numerație

● Împărțire cu rest a două numere naturale

Aplicația, prezentată în listingul 5.1, determină cîtul CIT și restul REST ale împărțirii întregi a numerelor N1 și N2.

```
10 PRINT "Impartirea cu rest a doua numere naturale nenule"
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40   GOSUB 3000 'Citeste si valideaza N1 si N2
50   GOSUB 1000 'Efectueaza impartirea cu rest
60   GOSUB 100 'Afiseaza N1,N2,CIT si REST
70   INPUT "Continuati ? (Y/N) : "
   C$
80 WEND
90 STOP
100 'Afiseaza N1,N2,CIT si REST
105 IF ER% = 0
   THEN PRINT N1;"=";N2"*";CIT;"+";REST
   ELSE PRINT "ER = ";ER%; " Tentativa de impartire la zero"
110 RETURN
1000 'Efectueaza impartirea cu rest N1=N2*CIT+REST
1001 'ER% = 0 Operatia s-a efectuat corect
1002 'ER% = -1 Tentativa de impartire la zero
1005 IF N2 <> 0
   THEN CIT=N1\N2 : REST N1 MOD N2 : ER%=0
   ELSE ER%=-1
1010 RETURN
MERGE "C1"
Ok
RUN
Impartirea cu rest a doua numere naturale nenule
N1 = 5
N2 = 2
5 = 2 * 2 + 1
Continuati ? (Y/N) : Y
N1 = 24
N2 = 0
ER: N2 numar eronat
N2 = 6
```

```

24 = 6 * 4 + 0
Continuati ? (Y/N) : N
Break in 90
Ok

```

Listing 5.1

Citirea și validarea deîmpărțitului N_1 și împărțitorului N_2 se realizează în rutina C1 (anexa A), prezentată în capitolul 4, figura 4.3.a, listîngul 4.9. Prin modul de realizare, aceasta obligă utilizatorul să tasteze numai numere corecte din punctul de vedere al problemei date.

Determinarea valorilor variabilelor CIT și REST este asigurată prin rutina R1, listîngul 5.1, liniile 1000-1010. Rutina verifică valoarea împărțitorului și, în cazul în care acesta este 0, returnează valoarea $ER\% = -1$ (tentativa de împărțire la 0), fără a mai efectua operația. Schema logică a rutinei R1 este prezentată în figura 5.2. Rutina nu afectează valorile variabilelor N_1 și N_2 .

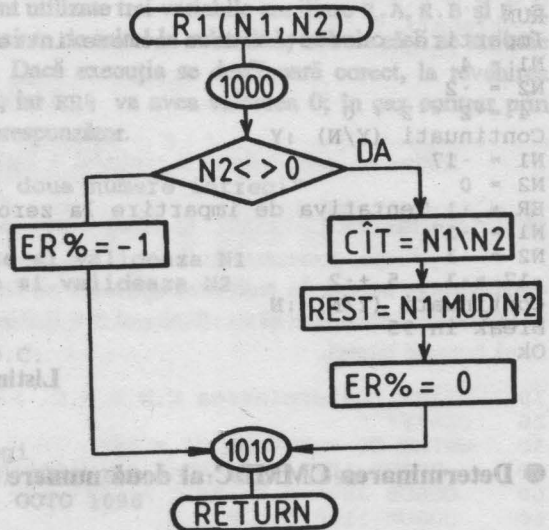


Fig. 5.2

● Împărțirea cu rest a două numere întregi

Această aplicație este o generalizare a celei anterioare prezentate. În realizarea aplicației s-au utilizat rutinele R1 (anexa A) și C2.

Rutina C2, listîngul 5.2, liniile 3040-3055, asigură citirea și validarea unui număr întreg. În cazul tastării unui număr corect, acesta este oferit programului principal prin variabila de ieșire C.A. Deoarece aceasta este o variabilă de manevră a rutinei C2 (care este automat alterată la următorul salt la subrutină), valoarea ei este atribuită variabilelor N_1 (după primul salt la subrutină <linia 40>), respectiv N_2 (după al doilea salt la subrutină <linia 50>).

```

10 PRINT "Impartirea cu rest a doua numere intregi"
20 CŞ="Y"
30 WHILE CŞ = "Y" OR CŞ = "y"
40 GOSUB 3040 : N1=C.A'Citeste si valideaza N1
50 GOSUB 3040 : N2=C.A'Citeste si valideaza N2
60 GOSUB 1000'Efectueaza impartirea
70 GOSUB 100' Afisare
80 INPUT "Continuati ? (Y/N) :", CŞ
90 WEND
95 STOP
3040 'Citeste si valideaza un numar intreg
3045 INPUT "N = ", C.A
3050 WHILE C.A <> INT(C.A) :

```

5. Aplicații

```

PRINT "ER: numar eronat" :
INPUT "N = ",C.A :
WEND
3055 RETURN
MERGE "R1"
Ok
RUN
Impartirea cu rest a doua numere intregi
N1 = 4
N2 = -2
4 = -2 * -2 + 0
Continuati (Y/N) :Y
N1 = -17
N2 = 0
ER = -1 Tentativa de impartire la zero
N1 = -17
N2 = -3
-17 = -3 * 5 + 2
Continuati (Y/N) :N
Break in 95
Ok
    
```

Lișting 5.2

● Determinarea CMMDC al două numere întregi

Aplicația permite determinarea celui mai mare divizor comun al două numere întregi și este utilă, alături de aplicațiile următoare, mai ales elevilor din clasele 5 - 7 în lucrul cu fracții.

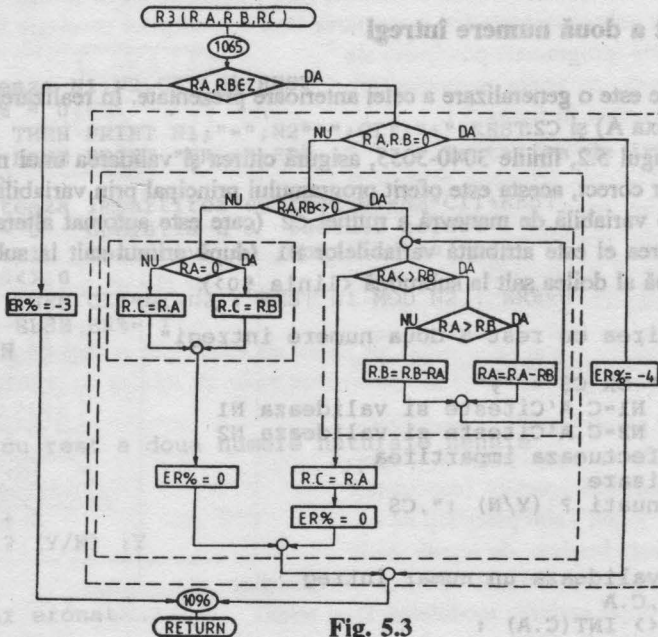


Fig. 5.3

În realizarea aplicației au fost utilizate rutinele C2 (anexa A), pentru introducerea și validarea variabilelor de intrare, R3 pentru determinarea CMMDC al două numere întregi și S1 pentru afișarea soluției.

Rutina R3 este prezentată în listingul 5.3, liniile 1065 - 1096. Schema logică a rutinei este ilustrată în figura 5.3. La determinarea CMMDC sînt utilizate trei variabile auxiliare R.A, R.B și R.C. Variabilelor R.A și R.B, considerate de intrare înainte de saltul la subrutină, trebuie să li se atribuie valorile variabilelor al căror CMMDC îl căutăm. Dacă execuția se desfășoară corect, la revenirea din subrutină variabila R.C va conține CMMDC, iar ER% va avea valoarea 0; în caz contrar prin variabila ER% se va transmite un cod de eroare corespunzător.

```

10 PRINT "Determinarea C.M.M.D.C. al doua numere intregi"
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40 GOSUB 3040:R.A=ABS(C.A) 'Citeste si valideaza N1
50 GOSUB 3040:R.B=ABS(C.A) 'Citeste si valideaza N2
60 GOSUB 1065 'C.M.M.D.C.
70 IF ER% = 0 THEN CMMDC=R.C
80 GOSUB 5000 'Afisare C.M.M.D.C.
90 INPUT "Continuati ? (Y/N) :";C$
100 WEND
110 STOP
1065 'C.M.M.D.C. al doua numere intregi
1070 IF R.A<>INT(R.A) OR R.B<>INT(R.B) THEN ER%=-3 : GOTO 1096
1071 IF R.A=0 AND R.B=0 THEN ER%=-4 : GOTO 1096
1075 IF R.A<>0 AND R.B<>0
    THENGOSUB 1087
    ELSEGOSUB 1094
1080 RETURN
1086 '
1087 WHILE R.A<>R.B
1088     IF R.A > R.B
        THENR.A=R.A-R.B
        ELSER.B=R.B-R.A
1089 WEND
1090 R.C=R.A
1091 ER%=0
1092 RETURN
1093 '
1094 IF R.A = 0
    THENR.C=R.B
    ELSER.C=R.A
1095 ER%=0
1096 RETURN
MERGE "C2"
Ok
MERGE "S1"
Ok
RUN
Determinarea C.M.M.D.C. al doua numere intregi
N = 24
N = 18
C.M.M.D.C. = 6
Continuati ? (Y/N) :Y
N = 0

```

5. Aplicații

```

N = 0
ER = -4
Continuati ? (Y/N) :Y
N = -12
N = 15
C.M.M.D.C. = 3
Continuati ? (Y/N) :N
Break in 110
Ok
    
```

Listing 5.3

Rutina S1 este definită în capitolul 4, listingul 4.6.

● Determinarea CMMMC al două numere întregi

Aplicația, prezentată în listingul 5.4, permite determinarea celui mai mic multiplu comun al două numere întregi. În realizarea aplicației au fost utilizate rutinele C2, R3, S3 (anexa A), definite în aplicațiile anterioare, și R4 (listingul 5.4, liniile 1100 - 1115) care determină efectiv CMMMC al două numere întregi.

```

10 PRINT "Determinarea C.M.M.M.C. al doua numere intregi"
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40   GOSUB 3040:R.A=ABS(C.A)'Citeste si valideaza N1
50   GOSUB 3040:R.B=ABS(C.A)'Citeste si valideaza N2
60   GOSUB 1100'C.M.M.M.C.
70   IF ER% = 0 THEN CMMMC=R.C
80   GOSUB 5020'Afisare C.M.M.M.C.
90   INPUT "Continuati ? (Y/N) :","C$
100 WEND
110 STOP
1100 'C.M.M.M.C. al doua numere intregi
1105 R.A1=R.A: R.B1=R.B : GOSUB 1065 'C.M.-
M.D.C.
1110 IF ER% = 0 THEN R.C=R.A1/R.C*R.B1
1115 RETURN
MERGE "C2"
Ok
MERGE "R3"
Ok
MERGE "S2"
Ok
RUN
    
```

Determinarea C.M.M.M.C. al doua numere intregi

```

N = 6
N = 4
C.M.M.M.C. = 12
Continuati ? (Y/N) :Y
N = 84
N = 20
C.M.M.M.C. = 420
Continuati ? (Y/N) :N
Break in 110
Ok
    
```

Listing 5.4

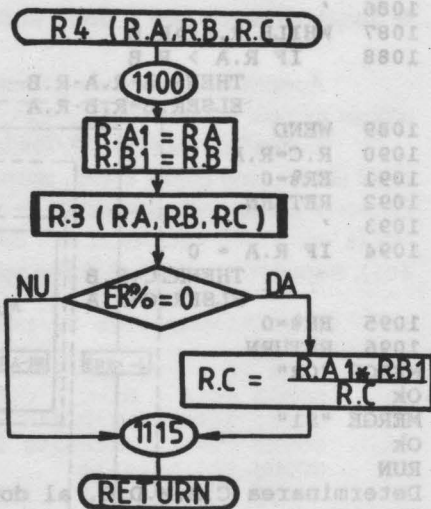


Fig. 5.4

Rutina R4, a cărei schemă logică este prezentată în figura 5.4, utilizează variabilele de intrare R.A și R.B, de iesire R.C (conține valoarea CMMMC), precum și variabilele auxiliare R.A1 și R.A2 cu rol de a 'proteja' (de a păstra nemodificate) valorile variabilelor R.A și R.B utilizate și de rutina R3 apelată de aceasta.

● Determinarea CMMDC al maxim 20 de numere întregi

Aplicația constituie o generalizare a celor prezentate în 5.1.3 pentru un șir de maxim 20 de numere întregi. În cadrul acestei aplicații se definesc două noi rutine C3 și R5.

Rutina C3, listingul 5.5, liniile 3060 - 3095, permite citirea și validarea unui vector de maxim 20 de numere întregi. Valorile celor N numere corect tastate sînt depuse succesiv în vectorul A. Variabilele A și N sînt considerate globale.

```

10 PRINT "Determinarea C.M.M.D.C. al 2<=N<=20 numere intregi"
15 DIM A(20)
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40 GOSUB 3060 'Citeste N si A(N)
50 R.N=N : GOSUB 1120 'C.M.M.D.C
60 IF ER% = 0 THEN CMMDC=R.C
70 GOSUB 5000 'Afisare C.M.M.D.C
80 INPUT "Continuati ? (Y/N) :",C$
90 WEND
100 STOP
1120 'C.M.M.D.C AL 2<=N<=20 numere intregi
1125 IF R.N<2 OR R.N>20 THEN ER%=-5 : GOTO 1135
1126 R.E=0 : R.I=2
1127 WHILE R.E = 0 AND R.I <= R.N :
1128 R.E=A(R.I-1) :
1129 R.I=R.I+1
1128 WEND
1129 R.I=R.I-1
1130 IF R.E <> 0
1131 THEN R.A=ABS(A(R.I-1)) :
1132 R.B=ABS(A(0)) :
1133 GOSUB 1065 :
1134 R.I=2 :
1135 WHILE ER%=0 AND R.C<>1 AND R.I<=R.N :
1136 R.A=R.C :
1137 R.B=ABS(A(R.I-1)) :
1138 GOSUB 1065 :
1139 R.I=R.I+1 :
1140 WEND
1141 ELSE ER%=-4
1135 RETURN
3060 'Citeste 2<=N<=20 numere intregi in A(N)
3065 INPUT "Precizati N (2<=N<=20) : ",N
3066 WHILE INT(N) <> N OR N < 2 OR N > 20 :
3067 PRINT "ER : numar eronat" :
3068 INPUT "Precizati N (2<=N<=20) : ",N :
3069 WEND
3075 FOR C.I = 0 TO N-1

```

5. Aplicații

```

3080 PRINT "A(";C.I;") " ; :
      INPUT "= ",A(C.I)
3085 WHILE INT(A(C.I))<> A(C.I):
      PRINT "ER : numar eronat":
      PRINT "A(";C.I;") " ;
      INPUT "= ",A(C.I) :
      WEND
3090 NEXT
3095 RETURN
MERGE "R3"
Ok
MERGE "S1"
Ok
RUN
Determinarea C.M.M.D.C. al 2<=N<=20
numere intregi
Precizati N (2<=N<=20) : 3
  
```

```

A( 0 ) = 0
A( 1 ) = 12
A( 2 ) = 6
C.M.M.D.C.= 6
Continuati ? (Y/N):Y
Precizati N (2<=N<=20) : 6
A( 0 ) = 21
A( 1 ) = 63
A( 2 ) = 147
A( 3 ) = 231
A( 4 ) = 105
A( 5 ) = 273
C.M.M.D.C.= 21
Continuati ? (Y/N) :N
Break in 100
Ok
  
```

Listing 5.5

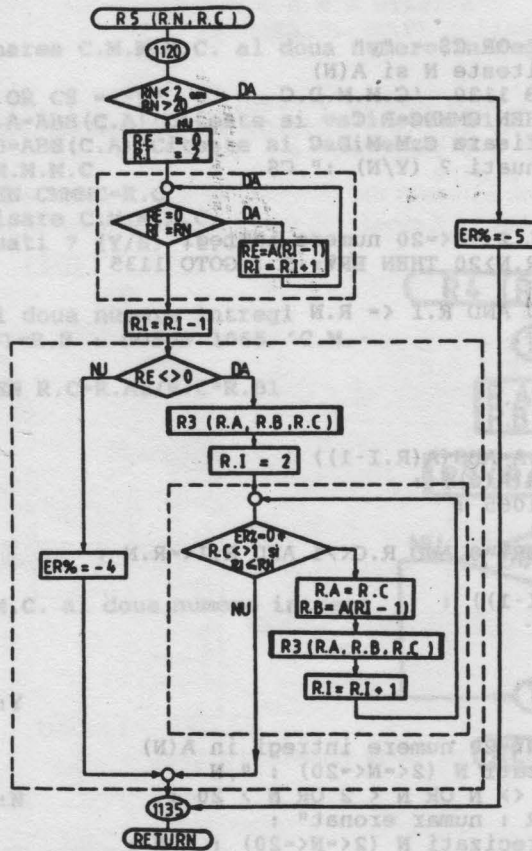


Fig. 5.5

Rutina R5, listingul 5.5, liniile 1120 - 1135, determină CMMDC al celor N numere întregi conținute în vectorul A. Schema logică a rutinei este prezentată în figura 5.5. Valoarea CMMDC determinată este atribuită variabilei R.C.

● Determinarea CMMMC al maxim 20 de numere întregi

Aplicația este o generalizare a celei prezentate în 5.1.4. În realizarea sa s-au utilizat rutinele C3, R3, S2 (anexa A) și R7 definită în cadrul acestei aplicații.

Rutina R7 permite determinarea CMMMC al celor N numere întregi depuse în vectorul A. Schema logică a rutinei este ilustrată în figura 5.6.

Aplicația este prezentată în listingul 5.6 împreună cu textul sursă al rutinei R7 (liniile 1180 - 1280).

```

10 PRINT "Determinarea C.M.M.M.C. al 2<=N<=20 numere intregi"
15 DIM A(20)
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40   GOSUB 3060'Citeste N si A(N)
50   R.N=N : GOSUB 1180'C.M.M.M.C
60   IF ER% = 0 THEN CMMMC=R.C
70   GOSUB 5020'Afisare C.M.M.M.C.
80   INPUT "Continuati ? (Y/N) :",C$
90 WEND
100 STOP
1180 'C.M.M.M.C. al 2<=N<=20 numere intregi
1185 IF R.N<2 OR R.N>20 THEN ER%=-5 : GOTO 1275
1186 R.I=1
1187 WHILE A(R.I-1)<>0 AND R.I<=R.N :
1188   R.I=R.I+1 :
1189 WEND
1190 IF R.I <= R.N THEN R.MC=0 : GOTO 1275
1195 R.A=ABS(A(0)) : R.B=ABS(A(1)) : GOSUB 1065
1200 IF ER% <> 0 THEN GOTO 1275
1205 R.DCA=R.C
1210 R.MCA=ABS(A(0))/R.DCA*ABS(A(1))
1215 R.I=3 : R.MC=R.MCA
1220 WHILE R.I <= R.N AND ER% = 0
1225   R.A=R.DCA : R.B=ABS(A(R.I-1)) : GOSUB 1065
1230   IF ER% <> 0 THEN GOTO 1270
1235   R.MCB=R.DCA/R.C*ABS(A(R.I-1))
1240   R.DCA=R.C
1245   R.A=R.MCA : R.B=R.MCB : GOSUB 1065
1250   IF ER% <> 0 THEN GOTO 1270
1255   R.MC=R.MCA/R.C*R.MCB
1260   R.MCA=R.MC
1265   R.I=R.I+1
1270 WEND
1275 IF ER% = 0 THEN R.C=R.MC
1280 RETURN
MERGE "C3"
Ok
MERGE "R3"

```

5. Aplicații

OK
 MERGE "S2"
 OK
 RUN
 Determinarea C.M.M.M.C. al $2 \leq N \leq 20$ numere intregi
 Precizati N ($2 \leq N \leq 20$) : 4

A(0) = 2
 A(1) = 3
 A(2) = 4
 A(3) = 5
 C.M.M.D.C. = 60
 Continuați ? (Y/N) : Y
 Precizati N ($2 \leq N \leq 20$) : 5

A(0) = 10
 A(1) = 21
 A(2) = 35
 A(3) = 3
 A(4) = 42
 C.M.M.D.C. = 210
 Continuați ? (Y/N) : N
 Break in 100

Ok

Listing 5.6

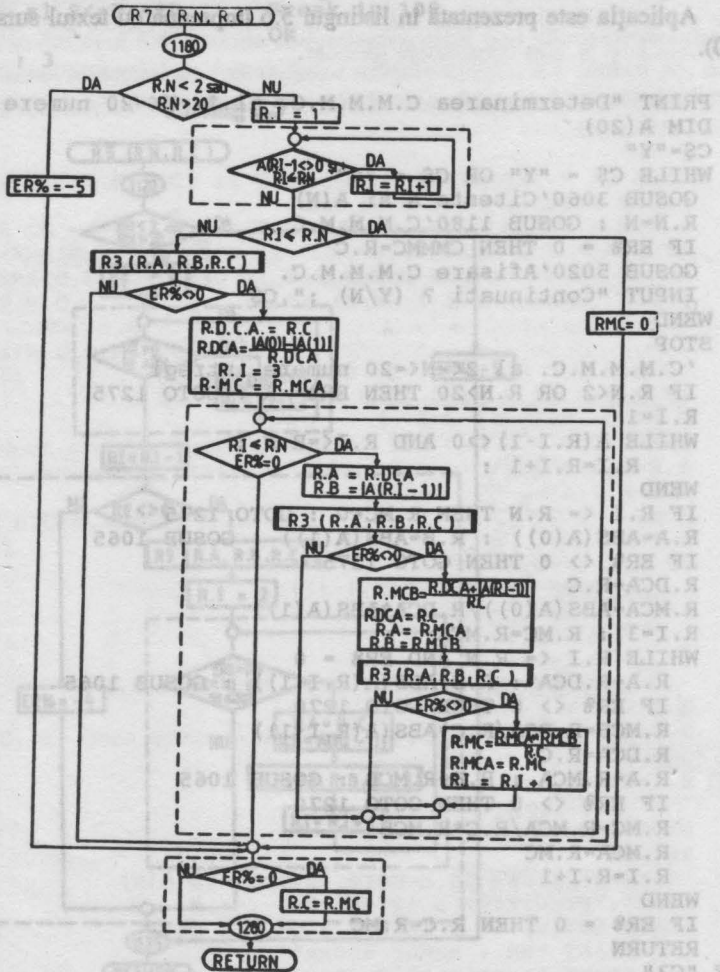


Fig. 5.6

Observație. Utilizarea acestei aplicații în determinarea CMMMC al mai multor numere întregi trebuie făcută cu atenție deoarece, dacă se lucrează cu numere relativ mari sau cu mai multe numere prime între ele, în procesul de multiplicare se poate depăși capacitatea maximă de reprezentare internă a numerelor întregi și, ca atare, rezultatul va fi eronat.

● Generarea de numere prime

Aplicația permite determinarea numerelor prime mai mari sau egale cu 2 și mai mici sau egale cu un număr N (maxim 1000).

În cadrul acestei aplicații este definită rutina R8, listingul 5.7, liniile 1300 - 1375, care, utilizând algoritmul lui Euclid, determină și depune în vectorul PRIM toate cele NRPRIM numere prime ce aparțin intervalului [2, R.N]. Rutina utilizează variabila de intrare R.N și variabilele auxiliare R.I, R.K, R.NF, R.N1 și vectorul R.AS care, după determinarea numerelor prime aparținând intervalului dat, este șters (zona de memorie ocupată de acesta este disponibilizată). Schema logică a rutinei este prezentată în figura 5.7.

```

10 PRINT "Genereaza numerele prime <= N <= 1000"
15 DIM PRIM(168)
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40 GOSUB 3040 : N=C.A'Citeste si valideaza N
50 R.N=N : GOSUB 1300'Generare
60 GOSUB 100'Afisare
70 INPUT "Continuati ? (Y/N) :",C$
80 WEND
90 STOP
100 '
105 IF ER% = 0
    THEN PRINT "Numerele prime <=";N;" sint : " :
    FOR I = 0 TO NRPRIM-1 STEP 12 :
    GOSUB 115 :
    NEXT :
    ELSE PRINT "ER = ";ER%
110 RETURN
115 '
120 IF I+10 > NRPRIM-1
    THEN J1=NRPRIM-1
    ELSE J1=I+10
125 FOR J = I TO J1 :
    PRINT USING "####";PRIM(J);:
NEXT
130 IF I+11 <= NRPRIM-1
    THEN PRINT USING "####";PRIM(I+11)
    ELSE PRINT
135 RETURN
1300 'Determina numerele prime 2<=R.N<=1000
1305 IF R.N<2 OR R.N>1000 THEN ER%=-6 : GOTO 1375
1310 DIM R.AS(998)
1315 FOR R.I=2 TO R.N
1316 R.AS(R.I-2)=R.I
1317 NEXT

```

5. Aplicații

```

1320 R.NF=INT(SQR(R.N))
1325 R.K=2
1330 WHILE R.K <= R.NF
1335 IF R.AS(R.K-2) <> 0
      THEN R.N1=R.AS(R.K-2)*2 :
WHILE R.N1 <= R.N :
      R.AS(R.N1-2)=0 :
      R.N1=R.N1+R.AS(R.K-2) :
WEND
1336 R.K=R.K+1
1340 WEND
1345 R.K=0
1350 FOR R.I=0 TO R.N-2
1351 IF R.AS(R.I) <> 0
      THEN PRIM(R.K)=R.AS(R.I) :
      R.K=R.K+1
1355 NEXT
1360 NRPRIM=R.K
1365 ERASE R.AS
1370 ER% = 0
1375 RETURN
MERGE "C2"
Ok
RUN

```

Genereaza numerele prime <= N <= 1000
N = 50
Numerele prime <= 50 sint :
 2 3 5 7 11 13 17 19 23 29 31 37
 41 43 47

Continuati ? (Y/N) : Y
N = 100
Numerele prime <= 100 sint :
 2 3 5 7 11 13 17 19 23 29 31 37
 41 43 47 53 59 61 67 71 73 79 83 89
 97

Continuati ? (Y/N) : Y
N = 1000
Numerele prime <= 1000 sint :
 2 3 5 7 11 13 17 19 23 29 31 37
 41 43 47 53 59 61 67 71 73 79 83 89
 97 101 103 107 109 113 127 131 137 139 149 151
 157 163 167 173 179 181 191 193 197 199 211 223
 227 229 233 239 241 251 257 263 269 271 277 281
 283 293 307 311 313 317 331 337 347 349 353 359
 367 373 379 383 389 397 401 409 419 421 431 433
 439 443 449 457 461 463 467 479 487 491 499 503
 509 521 523 541 547 557 563 569 571 577 587 593
 599 601 607 613 617 619 631 641 643 647 653 659
 661 673 677 683 691 701 709 719 727 733 739 743
 751 757 761 769 773 787 797 809 811 821 823 827
 829 839 853 857 859 863 877 881 883 887 907 911
 919 929 937 941 947 953 967 971 977 983 991 997

Continuati ? (Y/N) : N
Break in 90
 Ok

Listing 5.7

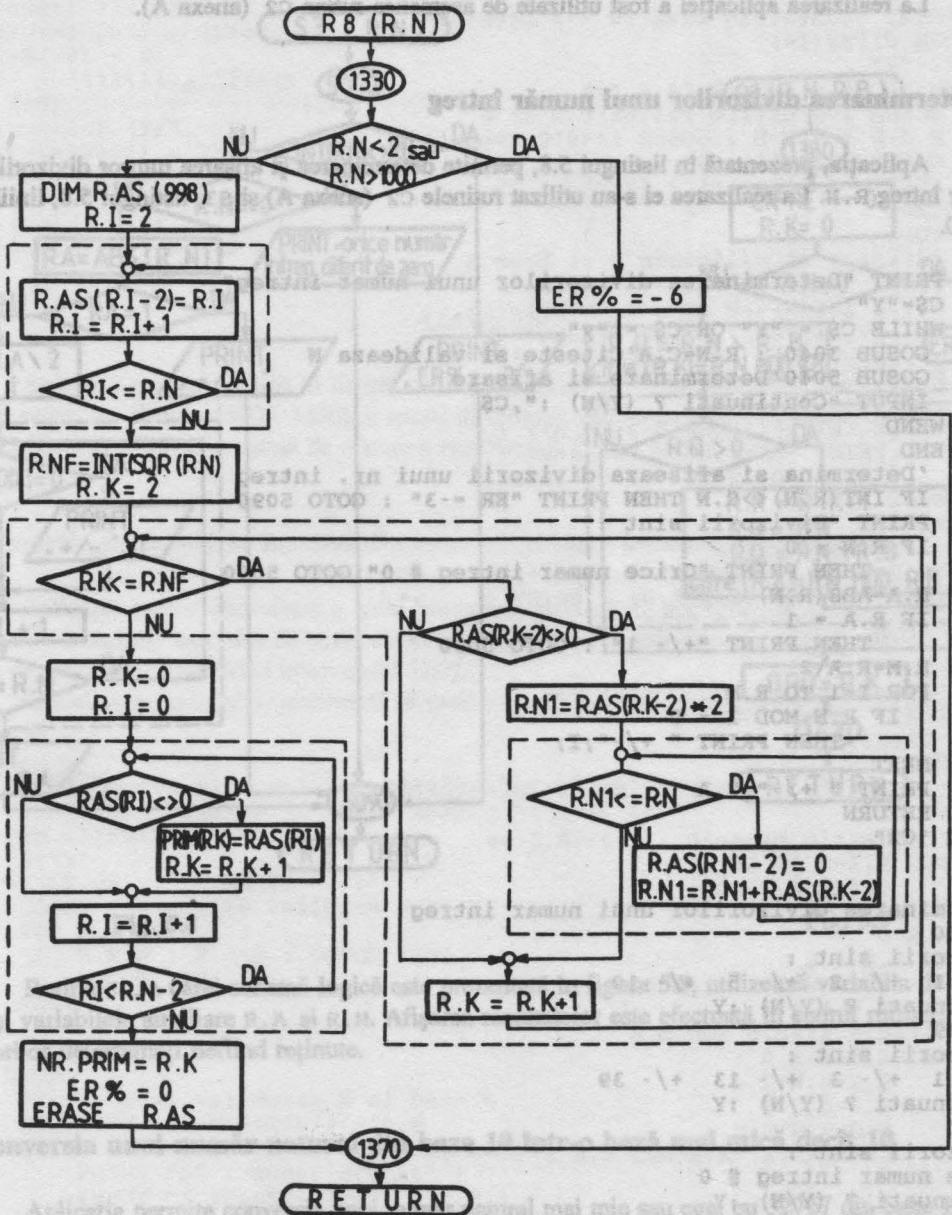


Fig. 5.7

La realizarea aplicației a fost utilizate de asemenea rutina C2 (anexa A).

● Determinarea divizorilor unui număr întreg

Aplicația, prezentată în listingul 5.8, permite determinarea și afișarea tuturor divizorilor unui număr întreg R.N. La realizarea ei s-au utilizat rutinele C2 (anexa A) și S3, listingul 5.8, liniile 5040 - 5090.

```

10 PRINT "Determinarea divizorilor unui numar intreg"
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40   GOSUB 3040 : R.N=C.A'Citeste si valideaza N
50   GOSUB 5040'Determinare si afisare
60   INPUT "Continuati ? (Y/N) :",C$
70 WEND
80 END
5040 'Determina si afiseaza divizorii unui nr. intreg
5045 IF INT(R.N)<>R.N THEN PRINT "ER --3" : GOTO 5090
5050 PRINT "Divizorii sint :"
5055 IF R.N = 0
    THEN PRINT "Orice numar intreg # 0":GOTO 5090
5060 R.A=ABS(R.N)
5065 IF R.A = 1
    THEN PRINT "+/- 1" : GOTO 5090
5070 R.M=R.A\2
5075 FOR I=1 TO R.M
5076   IF R.N MOD I = 0
    THEN PRINT " +/-";I;
5080 NEXT
5085 PRINT " +/-";R.A
5090 RETURN
MERGE "C2"
Ok
RUN

```

Determinarea divizorilor unui numar intreg

N = 10

Divizorii sint :

+/- 1 +/- 2 +/- 5 +/- 10

Continuati ? (Y/N) :Y

N = 39

Divizorii sint :

+/- 1 +/- 3 +/- 13 +/- 39

Continuati ? (Y/N) :Y

N = 0

Divizorii sint :

Orice numar intreg # 0

Continuati ? (Y/N) :Y

N = -41

Divizorii sint :

+/- 1 +/- 41

Continuati ? (Y/N) :N

Ok

Listing 5.8

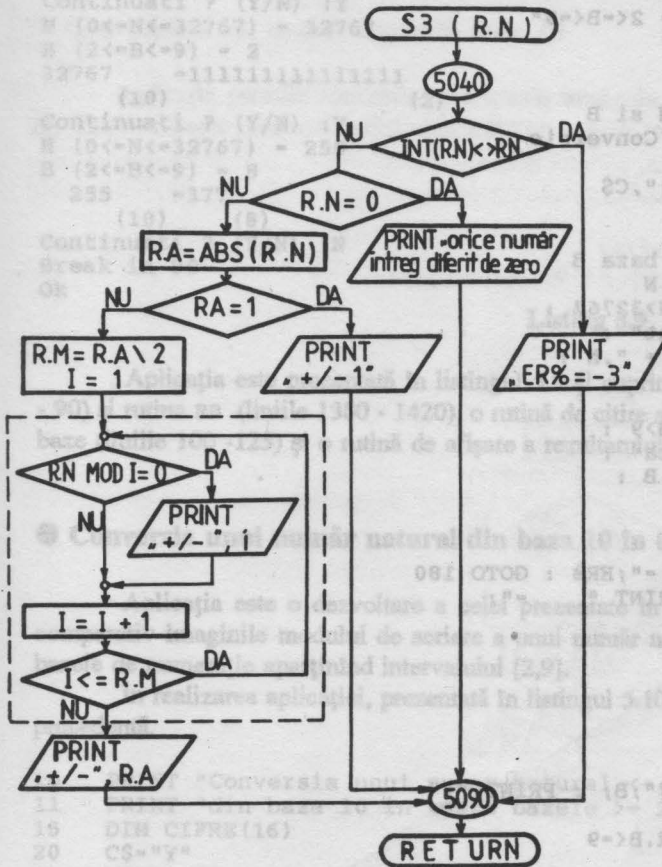


Fig. 5.8

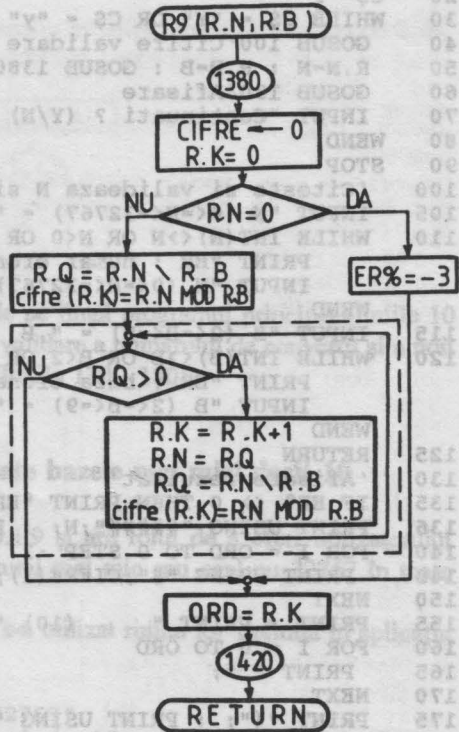


Fig. 5.9

Rutina S3, a cărei schemă logică este prezentată în figura 5.8, utilizează variabila de intrare R.N și variabilele auxiliare R.A și R.M. Afișarea rezultatelor este efectuată în cadrul rutinei, valorile divizorilor determinați nefiind reținute.

● Conversia unui număr natural din baza 10 într-o bază mai mică decât 10

Aplicația permite conversia unui număr natural mai mic sau egal cu 32767 din baza 10 într-o bază aparținând intervalului [2,9]. Conversia este realizată cu rutina R9 a cărei schemă logică este prezentată în figura 5.9.

Rutina R9, listingul 5.9, liniile 1380 - 1420, utilizează variabilele de intrare R.N (numărul de convertit) și R.B (noua bază), rezultatul (sub forma unui șir de cifre) fiind depus în vectorul CIFRE. Ordinul de mărime (numărul de cifre) al rezultatului este depus în variabila ORD.

5. Aplicații

```

10 PRINT "Conversie baza 10 --> 2<=B<=9"
15 DIM CIFRE(16)
20 CŞ="Y"
30 WHILE CŞ = "Y" OR CŞ = "y"
40   GOSUB 100'Citire validare N si B
50   R.N=N : R.B=B : GOSUB 1380'Conversie
60   GOSUB 130'Afisare
70   INPUT "Continuati ? (Y/N) :",CŞ
80 WEND
90 STOP
100 'Citeste si valideaza N si baza B
105 INPUT "N (0<=N<=32767) = ",N
110 WHILE INT(N)<>N OR N<0 OR N>32767 :
    PRINT "ER : numar eronat" :
    INPUT "N (0<=N<=32767) = ",N :
WEND
115 INPUT "B (2<=B<=9) = ",B
120 WHILE INT(B)<>B OR B<2 OR B>9 :
    PRINT "ER : baza eronata" :
    INPUT "B (2<=B<=9) = ",B :
WEND
125 RETURN
130 'Afisare rezultat
135 IF ER% <> 0 THEN PRINT "ER =";ER% : GOTO 180
136 PRINT USING "#####;N; : PRINT " =";
140 FOR I = ORD TO 0 STEP -1
145   PRINT USING "#";CIFRE(I);
150 NEXT
155 PRINT : PRINT " (10) ";
160 FOR I = 0 TO ORD
165   PRINT " ";
170 NEXT
175 PRINT "("; : PRINT USING "#";B; : PRINT ")"
180 RETURN
1380 'Conversie baza 10 --> 2<=R.B<=9
1385 FOR R.I=0 TO 15 :
    CIFRE(R.I)=0 :
NEXT
1390 R.K=0
1395 IF R.N < 0
    THEN ER%=-3 : GOTO 1415
1400 R.Q=R.N\R.B
1405 CIFRE(R.K)=R.N MOD R.B
1410 WHILE R.Q > 0 :
    R.K=R.K+1 :
    R.N=R.Q :
    R.Q=R.N\R.B :
    CIFRE(R.K)=R.N MOD R.B :
WEND
1415 ORD=R.K
1420 RETURN
RUN
Conversie baza 10 --> 2<=B<=9
N (0<=N<=32767) = 15
B (2<=B<=9) = 2
15 =1111
(10) (2)

```

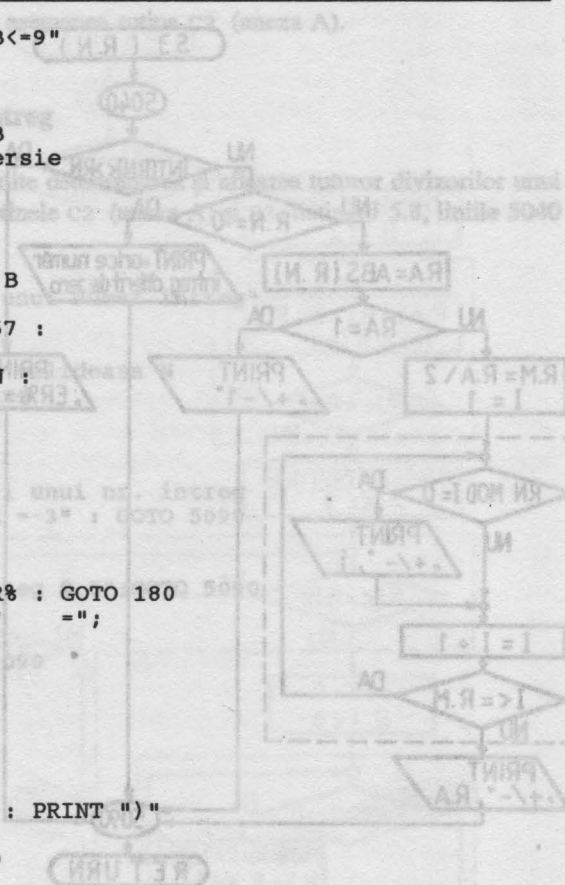


Fig. 2.8

```

Continuati ? (Y/N) :Y
N (0<=N<=32767) = 32767
B (2<=B<=9) = 2
32767 =1111111111111111

```

```

Continuati ? (Y/N) :Y
N (0<=N<=32767) = 255
B (2<=B<=9) = 8
255 =377

```

```

Continuati ? (Y/N) :N
Break in 90
Ok

```

Listing 5.9

Aplicația este prezentată în listingul 5.9 și cuprinde pe lângă programul principal (liniile 10 - 90) și rutina R9 (liniile 1380 - 1420), o rutină de citire și validare a numărului de convertit și a noii baze (liniile 100 -125) și o rutină de afișare a rezultatului (liniile 130 - 180).

● Conversia unui număr natural din baza 10 în toate bazele mai mici decât 10

Aplicația este o dezvoltare a celei prezentate în 5.1.9 și are rolul de a oferi utilizatorului comparativ imaginile modului de scriere a unui număr natural mai mic sau egal cu 32767 în toate bazele de numerație aparținând intervalului [2,9].

În realizarea aplicației, prezentată în listingul 5.10, s-a utilizat rutina R9 definită în aplicația precedentă.

```

10 PRINT "Conversia unui numar natural <= 32767"
11 PRINT "din baza 10 in toate bazele >= 2 si <= 9"
15 DIM CIFRE(16)
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40 GOSUB 100 'Citire validare N si B
50 FOR B = 2 TO 9 :
    R.N=N : R.B=B : GOSUB 1380 :
    GOSUB 130
60 NEXT
70 INPUT "Continuati ? (Y/N) :",C$
80 WEND
90 STOP
100 'Citeste si valideaza N si baza B
105 INPUT "N (0<=N<=32767) = ",N
110 WHILE INT(N)<>N OR N<0 OR N>32767 :
    PRINT "ER : numar eronat" :
    INPUT "N (0<=N<=32767) = ",N :
WEND
125 RETURN
130 'Afișare rezultat
135 IF ER% <> 0 THEN PRINT "ER =";ER% : GOTO 180
136 PRINT USING "#####";N; : PRINT " =";
140 FOR I = ORD TO 0 STEP -1
145 PRINT USING "#";CIFRE(I);

```

5. Aplicații

```

150 NEXT
155 PRINT : PRINT " (10) ";
160 FOR I = 0 TO ORD
165 PRINT " ";
170 NEXT
175 PRINT "(", : PRINT USING "#";B; : PRINT ")"
180 RETURN
MERGE "R9"
Ok
RUN

```

Conversia unui numar natural <= 32767
din baza 10 in toate bazele >= 2 si <= 9

```
N (0<=N<=32767) = 15
```

```

15 =1111 (2)
(10) (2)
15 =120 (3)
(10) (3)
15 =33 (4)
(10) (4)
15 =30 (5)
(10) (5)
15 =23 (6)
(10) (6)
15 =21 (7)
(10) (7)
15 =17 (8)
(10) (8)
15 =16 (9)
(10) (9)

```

```
Continuati ? (Y/N) :Y
```

```
N (0<=N<=32767) = 1283
```

```

15 =10100000011 (2)
(10) (2)
15 =1202112 (3)
(10) (3)
15 =110003 (4)
(10) (4)
15 =20113 (5)
(10) (5)
15 =5535 (6)
(10) (6)
15 =3512 (7)
(10) (7)
15 =2403 (8)
(10) (8)
15 =1675 (9)
(10) (9)

```

```
Continuati ? (Y/N) :N
```

```
Break in 99
```

```
Ok
```

Listing 5.10

● Conversia unui număr binar în baza 10

Aplicația permite conversia unui număr întreg de maxim 16 cifre binare din baza 2 în baza 10. Conversia este efectuată utilizând rutina R10 a cărei schemă logică este prezentată în figura 5.10.

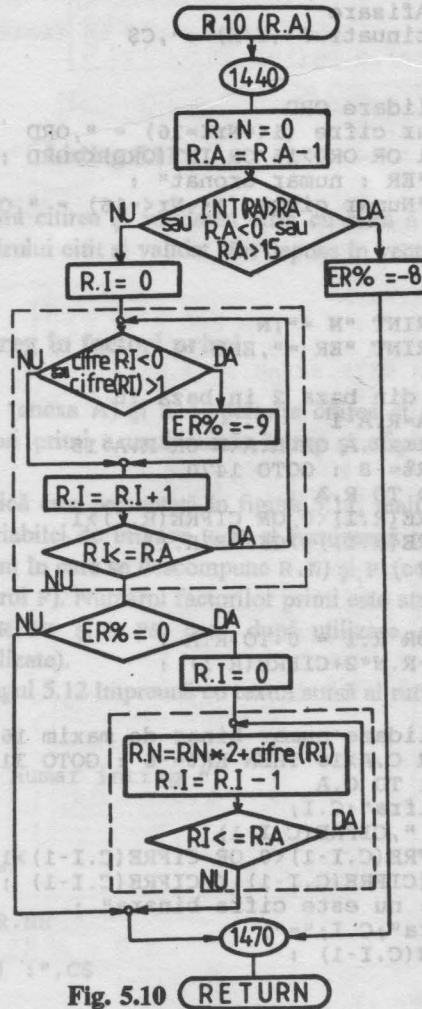


Fig. 5.10

Rutina R10 asigură conversia numărului ale cărui ORD cifre binare sînt depuse în vectorul CIFRE. Numărul întreg în baza 10 rezultat este atribuit variabilei R.A.

Aplicația este prezentată în listingul 5.11 și conține pe lângă rutina R10 (liniile 1440 - 1470) și textul sursă al rutinei C4 (liniile 3100 - 3130).

```

10 PRINT "Conversia in baza 10 a unui numar binar"
15 DIM CIFRE(16)
  
```

5. Aplicații

```

20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40   GOSUB 100 'Citire validare N
50   C.A=ORD : GOSUB 3100'Citire validare CIFRE
60   IF ER% = 0
       THEN R.A=ORD : GOSUB 1440 : N=R.N
70   GOSUB 135' Afisare
80   INPUT "Continuati ? (Y/N) : ", C$
90 WEND
95 STOP
100 'Citire validare ORD
105 INPUT "Numar cifre (1<=Nr<=16) = ", ORD
110 WHILE ORD<1 OR ORD>16 OR INT(ORD)<>ORD :
       PRINT "ER : numar eronat" :
       INPUT "Numar cifre (1<=Nr<=16) = ", ORD :
WEND
115 RETURN
135 ' Afisare
140 IF ER% = 0
       THEN PRINT "N = "; N
       ELSE PRINT "ER = ", ER%
145 RETURN
1440 'Conversie din baza 2 in baza 10.
1445 R.N=0 : R.A=R.A-1
1450 IF INT(R.A)<>R.A OR R.A<0 OR R.A>15
       THEN ER%=-8 : GOTO 1470
1455 FOR R.I = 0 TO R.A :
       IF CIFRE(R.I)<0 OR CIFRE(R.I)>1
       OR INT(CIFRE(R.I))<>CIFRE(R.I)
       THEN ER%=-9
1460 NEXT
1465 IF ER% = 0
       THEN FOR R.I = 0 TO R.A :
             R.N=R.N*2+CIFRE(R.I) :
           NEXT
1470 RETURN
3100 'Citire validare numar binar de maxim 16 cifre
3105 IF C.A<1 OR C.A>16 THEN ER%=-6 : GOTO 3130
3110 FOR C.I = 1 TO C.A
3111   PRINT "Cifra"; C.I;
3112   INPUT "= ", CIFRE(C.I-1)
3113   WHILE CIFRE(C.I-1)<0 OR CIFRE(C.I-1)>1
         OR INT(CIFRE(C.I-1))<>CIFRE(C.I-1) :
       PRINT "ER : nu este cifra binara" :
       PRINT "Cifra"; C.I; "= "; :
       INPUT CIFRE(C.I-1) :
WEND
3120 NEXT
3130 RETURN

```

RUN
 Conversia in baza 10 a unui numar binar

Numar cifre (1<=Nr<=16) = 4

Cifra 1 = 1

Cifra 2 = 1

Cifra 3 = 1

Cifra 4 = 1

N = 15

```

Continuati ? (Y/N) :Y
Numar cifre (1<=Nr<=16) = 6
Cifra 1 = 1
Cifra 2 = 0
Cifra 3 = 1
Cifra 4 = 0
Cifra 5 = 1
Cifra 6 = 0
N = 42
Continuati ? (Y/N) :N
Break in 95
Ok

```

Listing 5.11

Rutina C4 are rolul de a asigura citirea și validarea cifră cu cifră a unui număr binar de maxim 16 cifre. Cifrele binare ale numărului citit și validat sînt depuse în vectorul CIFRE.

● Descompunerea unui număr întreg în factori primi

Aplicația, utilizînd rutinele C2 (anexa A) și R11, permite citirea și validarea unui număr întreg, descompunerea acestuia în factori primi a unui număr întreg și afișarea ordonată și într-o formă intuitivă a acestora.

Rutina R11, a cărei schemă logică este prezentată în figura 5.11, realizează descompunerea în factori primi a numărului atribuit variabilei de intrare R.N și furnizează rezultatul în vectorii F (conținînd lista ordonată a factorilor primi în care se descompune R.N) și P (conținînd lista puterilor factorilor primi corespunzători din vectorul F). Numărul factorilor primi este atribuit variabilei R.NR. Rutina utilizează vectorii de manevră R.FS și R.PS care, după utilizare, sînt șterși (zonele de memorie afectate acestora sînt disponibilizate).

Aplicația este prezentată în listingul 5.12 împreună cu textul sursă al rutinei R11 (liniile 1480 - 1565).

```

10 PRINT "Descompunerea unui numar intreg "
11 PRINT "in factori primi"
15 DIM F(100),P(100)
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40 GOSUB 3040 : N=C.A
50 F.N=N : GOSUB 1480 : NR=R.NR
60 GOSUB 100
70 INPUT "Continuati ? (Y/N) :",C$
80 WEND
90 STOP
100 'Afișare
105 IF ER% = 0
    THEN GOSUB 115
    ELSE PRINT "ER = ";ER%
110 RETURN
115
120 PRINT "N = ";
125 IF N=0

```

5. Aplicații

```

20 THEN PRINT NR
30 ELSE FOR I=0 TO NR-1:
40 PRINT F(I);"^";P(I);"*";:
50 NEXT:
60 PRINT F(NR);"^";P(I)
130 PRINT
135 RETURN
1480 'Descompune un numar intreg R.N in factori primi
1485 DIM R.FS(100),R.PS(100)
1490 R.NR=0
1495 FOR R.I = 0 TO 100 :
1500 R.FS(R.I)=0 :
1510 R.PS(R.I)=0
1500 NEXT
1505 IF R.N<INT(R.N) THEN ER%-3 : GOTO 1560
1510 IF R.N=0 THEN GOTO 1555
1515 R.N=ABS(R.N)
1520 R.I=2
1525 WHILE R.I<=R.N
1530 R.Q=R.N\R.I : R.R=R.N MOD R.I
1535 IF R.R=0 AND R.Q<>0
1540 THEN R.PS(R.NR)=R.PS(R.NR)+1:
1545 R.FS(R.NR)=R.I:
1550 R.R=R.Q MOD R.I:
1555 R.N=R.Q\R.Q:R.Q=R.N\R.I:
1560 WHILE R.R=0 AND R.Q<>0:
1565 R.PS(R.NR)=R.PS(R.NR)+1:
1570 R.R=R.Q MOD R.I:
1575 R.N=R.Q\R.Q:R.Q=R.N\R.I:
1580 WEND:
1585 R.NR=R.NR+1
1590 R.I=R.I+1 : WEND : R.NR=R.NR-1
1595 FOR R.I = 0 TO R.NR :
1600 F(R.I)=R.FS(R.I) :
1605 P(R.I)=R.PS(R.I)
1610 NEXT
1615 ER%-0
1620 ERASE R.FS,R.PS
1625 RETURN
MERGE "C2"
Ok
RUN
Descompunerea unui numar intreg
in factori primi
N = 36
N = 2 * 2 * 3 * 3
Continuati ? (Y/N) : Y
N = 256
N = 2 * 2 * 8
Continuati ? (Y/N) : N
Break in 90
Ok

```

Listing 5.12

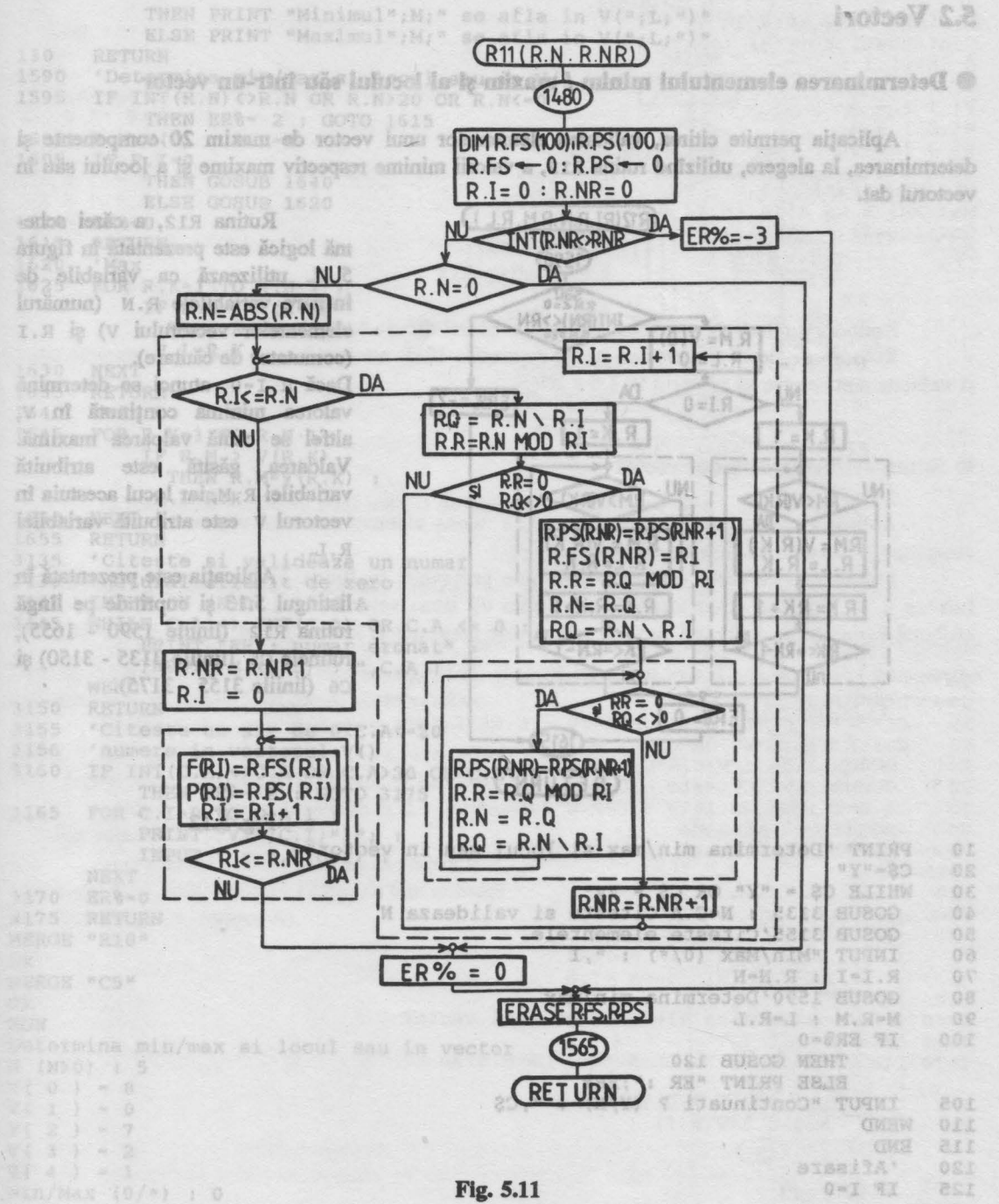
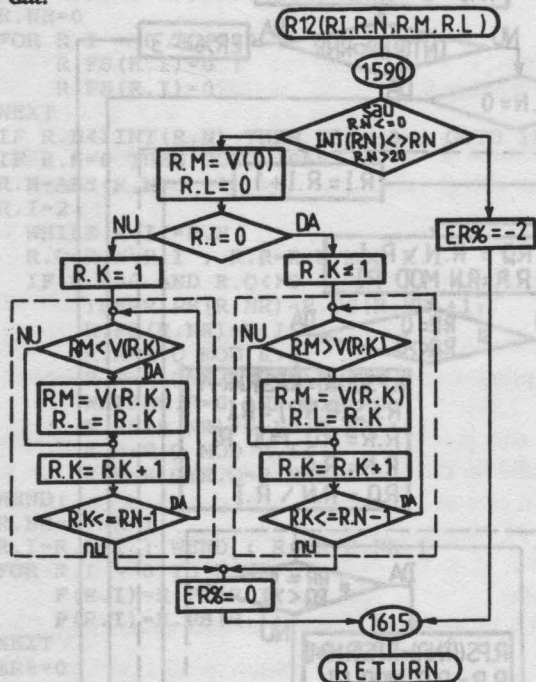


Fig. 5.11

5.2 Vectori

● Determinarea elementului minim / maxim și al locului său într-un vector

Aplicația permite citirea, validarea elementelor unui vector de maxim 20 componente și determinarea, la alegere, utilizând rutina R12, a valorii minime respectiv maxime și a locului său în vectorul dat.



Rutina R12, a cărei schemă logică este prezentată în figura 5.12, utilizează ca variabile de intrare variabilele R.N (numărul elementelor vectorului V) și R.I (comutator de căutare).

Dacă R.I=0 atunci se determină valoarea minimă conținută în V, altfel se caută valoarea maximă. Valoarea găsită este atribuită variabilei R.M, iar locul acestuia în vectorul V este atribuită variabilei R.L.

Aplicația este prezentată în listingul 5.13 și cuprinde pe lângă rutina R12 (liniile 1590 - 1655), rutinele C5 (liniile 3135 - 3150) și C6 (liniile 3155 - 3175).

Fig. 5.12

```

10 PRINT "Determina min/max si locul sau in vector"
20 C$="Y"
30 WHILE C$ = "Y" OR C$ = "y"
40 GOSUB 3135 : N=C.A'Citeste si valideaza N
50 GOSUB 3155'Citeste elementele
60 INPUT "Min/Max (0/*) : ",I
70 R.I=I : R.N=N
80 GOSUB 1590'Determina min/max
90 M=R.M : L=R.L
100 IF ER%=0
    THEN GOSUB 120
    ELSE PRINT "ER : ";ER%
105 INPUT "Continuati ? (Y/N) : ",C$
110 WEND
115 END
120 'Afisare
125 IF I=0
  
```

```

        THEN PRINT "Minimul";M;" se afla in V(";L;)"
        ELSE PRINT "Maximul";M;" se afla in V(";L;)"
130   RETURN
1590  'Determina min/max si locul sau in V()
1595  IF INT(R.N) <> R.N OR R.N > 20 OR R.N <= 0
        THEN ER% = -2 : GOTO 1615
1600  R.M = V(0) : R.L = 0
1605  IF R.I = 0
        THEN GOSUB 1640
        ELSE GOSUB 1620
1610  ER% = 0
1615  RETURN
1620  'Max
1625  FOR R.K = 1 TO R.N - 1 :
        IF R.M < V(R.K)
            THEN R.M = V(R.K) :
            R.L = R.K
1630  NEXT
1635  RETURN
1640  'Min
1645  FOR R.K = 1 TO R.N - 1 :
        IF R.M > V(R.K)
            THEN R.M = V(R.K) :
            R.L = R.K
1650  NEXT
1655  RETURN
3135  'Citeste si valideaza un numar
3136  'natural diferit de zero
3140  INPUT "N (N>0) : ", C.A
3145  WHILE C.A <> INT(C.A) OR C.A <= 0 :
        PRINT "ER : numar eronat" :
        INPUT "N (N>0) : ", C.A :
WEND
3150  RETURN
3155  'Citeste un sir de 0 < C.A <= 20
3156  'numar in vectorul V()
3160  IF INT(C.A) <> C.A OR C.A > 20 OR C.A <= 0
        THEN ER% = -2 : GOTO 3175
3165  FOR C.I = 0 TO C.A - 1 :
        PRINT "V(";C.I;)" :
        INPUT " = ", V(C.I) :
NEXT
3170  ER% = 0
3175  RETURN
MERGE "R10"
Ok
MERGE "C5"
Ok
RUN
Determina min/max si locul sau in vector
N (N>0) : 5
V( 0 ) = 8
V( 1 ) = 0
V( 2 ) = 7
V( 3 ) = 2
V( 4 ) = 1
Min/Max (0/*) : 0

```

5. Aplicații

```
Minimul 0 se afla in V( 1 )
Continuati ? (Y/N) : Y
N (N>0) : 5
V( 0 ) = 8
V( 1 ) = 0
V( 2 ) = 7
V( 3 ) = 2
V( 4 ) = 1
Min/Max (0/*) : 1
Maximul 8 se afla in V( 0 )
Continuati ? (Y/N) : N
Ok
```

Listing 5.13

Rutina C5 citește și validează un număr natural diferit de 0.

Rutina C6 citește și validează un șir de maxim 20 de numere întregi. Cele C.A numere citite și validate sînt depuse în vectorul V.

● Suma elementelor unui vector

Aplicația determină utilizînd rutina R13 suma elementelor unui vector V de maxim 20 componente, precum și afișarea acestuia.

Rutina R13, a cărei schemă logică este prezentată în figura 5.13, utilizează variabila de intrare R.N (numărul componentelor vectorului V), suma valorilor elementelor vectorului V fiind atribuită variabilei R.S. Textul sursă al rutinei este prezentat în listingul 5.14, liniile 1660 - 1675.

```
10 PRINT "Insumarea elementelor unui vector"
20 DIM V(20)
30 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
40 WHILE C$ = "Y" OR C$ = "y"
50 GOSUB 3180 : N=C.A.'Citeste N
60 GOSUB 3155.'Citeste elementele
70 R.N=N : GOSUB 1660 : S=R.S.'Suma
80 GOSUB 120.'Afisare
90 INPUT "Continuati ? (Y/N) : ",C$
100 WEND
110 END
120 'Afisare
130 IF ER%<>0
    THEN PRINT "ER =" ;ER%
    ELSE PRINT "Suma =" ;S
140 RETURN
1660 'Determina suma elementelor unui vector V()
1665 R.S=0
1670 IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
    THEN ER%=-2
    ELSE FOR R.I=0 TO R.N-1 :
        R.S=R.S+V(R.I) :
    NEXT :
    ER%=0
1675 RETURN
MERGE "C6"
```

```

Ok
MERGE "C7"
Ok
RUN
Insumarea elementelor unui vector
N( 2 <= N <= 20 ) = 4
V( 0 ) = 7
V( 1 ) = -3
V( 2 ) = 6
V( 3 ) = 5
Suma = 15
Continuati ? (Y/N) : N
Ok

```

Listing 5.14

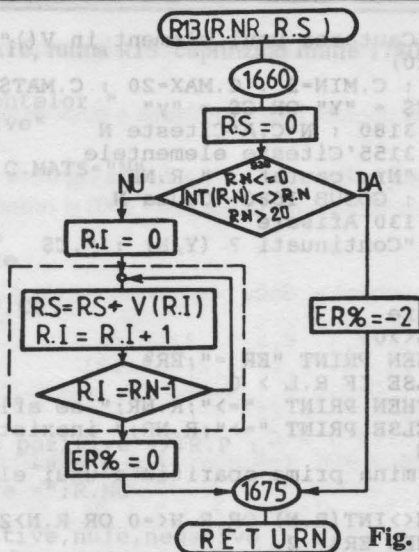


Fig. 5.13

În realizarea aplicației s-au utilizat de asemenea rutinele C6 și C7 (anexa A).

● Căutarea unei element într-un vector

Aplicația, prezentată în listingul 5.15, permite căutarea în vectorul V a unui element (număr întreg) oarecare și, în cazul în care acesta există, afișează locul său în V . Dacă elementul nu aparține vectorului V se afișează un mesaj corespunzător.

Căutarea elementului dorit $R.NR$ în vectorul V este realizată de rutina $R14$, a cărei schemă logică este prezentată în figura 5.14. Dacă $R.NR$ aparține vectorului V atunci $R.L$ conține poziția acestuia în V , altfel $R.L = -1$.

În realizarea aplicației în afara rutinei $R14$ (listingul 5.15, liniile 1680 - 1730), au fost utilizate rutinele $C6$ și $C7$ (anexa A).

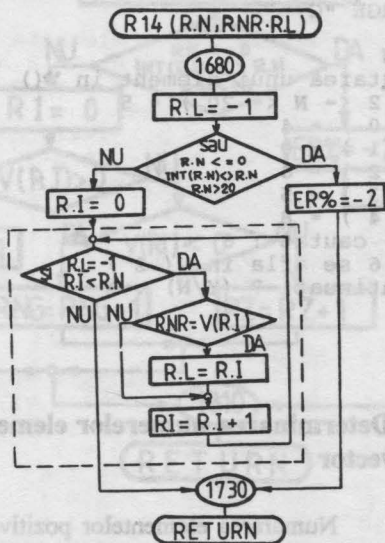


Fig. 5.14

5. Aplicații

```
10 PRINT "Cautarea unui element in V()"
20 DIM V(20)
30 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
40 WHILE C$ = "Y" OR C$ = "y"
50   GOSUB 3180 : N=C.A'Citeste N
60   GOSUB 3155'Citeste elementele
70   INPUT "Nr. cautat : ",R.NR
80   R.N=N : GOSUB 1680 'Cauta N
90   GOSUB 130'Afisare
100  INPUT "Continuati ? (Y/N) : ",C$
110 WEND
120 END
130 'Afisare
140 IF ER%<>0
    THEN PRINT "ER =" ;ER%
    ELSE IF R.L > 0
        THEN PRINT "=">" ;R.NR;" se afla in V(" ;R.L;" )
        ELSE PRINT "->" ;R.NR;" inexistent in V()
150 RETURN
1680 'Determina prima aparitie a unui element in V()
1685 R.L=-1
1690 IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
    THEN ER%=-2
    ELSE GOSUB 1700
1695 RETURN
1700 '
1705 R.I=0
1710 WHILE R.L=-1 AND R.I<R.N
1715   IF R.NR=V(R.I) THEN R.L=R.I
1720   R.I=R.I+1
1725 WEND
1730 RETURN
MERGE "C6"
Ok
MERGE "C7"
Ok
RUN
Cautarea unui element in V()
N( 2 <= N <= 20 ) = 5
V( 0 ) = 4
V( 1 ) = 0
V( 2 ) = 6
V( 3 ) = 5
V( 4 ) = 8
Nr. cautat : 6
=> 6 se afla in V( 2 )
Continuati ? (Y/N) : N
Ok
```

Listing 5.15

● Determinarea numerelor elementelor pozitive, nule și negative conținute într-un vector

Numărarea elementelor pozitive, nule și negative se realizează cu rutina R15 prezentată în figura 5.15. După revenirea din rutină numerele elementelor pozitive, negative și nule sint date de valorile variabilelor R.P, R.Z, respectiv R.NG.

Aplicația este prezentată în listingul 5.16, rutina R15 cuprinzând liniile 1740 - 1810.

```

10 PRINT "Determina numarul elementelor "
20 PRINT "pozitive,nule si negative"
30 DIM V(20)
40 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50 WHILE C$ = "Y" OR C$ = "y"
60   GOSUB 3180 : N=C.A'Citeste N
70   GOSUB 3155'Citeste elementele
80   R.N=N : GOSUB 1740'Determinare
90   GOSUB 130'Afisare
100  INPUT "Continuati ? (Y/N) : ",C$
110  WEND
120  END
130  'Afisare
140  IF ER%>0
      THEN PRINT "ER =" ;ER%
      ELSE PRINT "Nr. elemente pozitive =" ;R.P :
      PRINT "Nr. elemente nule =" ;R.Z :
      PRINT "Nr. elemente negative =" ;R.NG
150  RETURN
1740 'Detrmnare nr. elemente pozitive,nule,negative
1745 R.P=0 : R.Z=0 : R.NG=0
1780 IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
      THEN ER%=-2
      ELSE GOSUB 1790

```

```

1785 RETURN
1790 '
1795 FOR R.I=0 TO R.N-1
1800   IF V(R.I) > 0

```

```

      THEN R.P=R.P+1
      ELSE IF V(R.I) < 0
      THEN R.NG=R.NG+1
      ELSE R.Z=R.Z+1

```

```

1805 NEXT
1810 RETURN
MERGE "C6"
Ok
MERGE "C7"
Ok

```

```

RUN
Determina numarul elementelor
pozitive,nule si negative
N( 2 <= N <= 20 ) = 7

```

```

V( 0 ) = 4
V( 1 ) = 0
V( 2 ) = 6
V( 3 ) = -8
V( 4 ) = 8
V( 5 ) = -1
V( 6 ) = 5
Nr. elemente pozitive = 4
Nr. elemente nule = 1
Nr. elemente negative = 2
Continuati ? (Y/N) : N
Ok

```

Listing 5.16

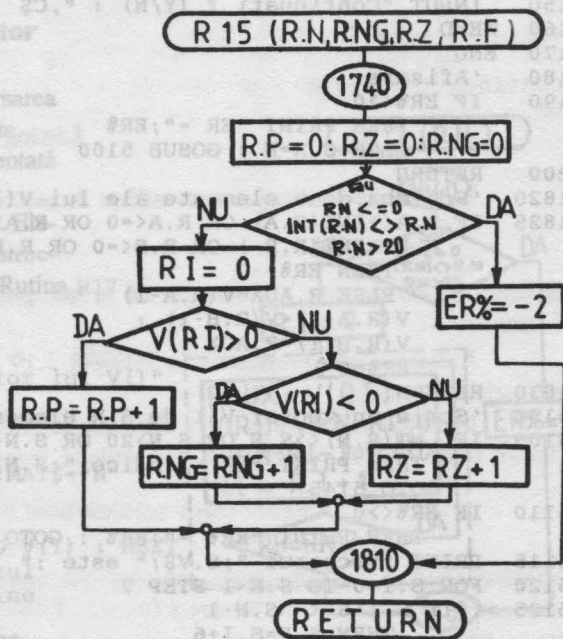


Fig. 5.15

● Interschimbarea a două elemente ale unui vector

Aplicația permite interschimbarea a două elemente ale vectorului V dat precizate prin poziția lor în acesta.

Prezentată în listingul 5.17, aplicația cuprinde pe lângă programul principal (liniile 10 - 170), rutina de afișare a vectorului (liniile 180 - 200) și rutinele R16 (liniile 1820 - 1830) și S4 (liniile 5100 - 5145).

```

10 PRINT "Schimba doua elemente inte ele"
20 DIM V(20)
30 DATA 10,6,7,3,-6,8,9,-11,6,24
35 DATA 4,11,-3,-8,0,-9,2,1,-1,21
40 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50 WHILE C$="Y" OR C$="Y"
60 GOSUB 3180 : N=C.A'Alege N
70 RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80 S.N=N : GOSUB 5100'Scrie vectorul
90 PRINT "Nr. element A ( 0 < Nr. <=";N;")";
100 INPUT " = ",R.A
110 PRINT "Nr. element B ( 0 < Nr. <=";N;")";
120 INPUT " = ",R.B
130 R.N=N : GOSUB 1820'Schimba elemente
140 GOSUB 180'Afizare
150 INPUT "Continuati ? (Y/N) : ",C$
160 WEND
170 END
180 'Afizare
190 IF ER%>0
    THEN PRINT "ER =" ;ER%
    ELSE S.N=N : GOSUB 5100
200 RETURN
1820 'Schimba doua elemente ale lui V() intre ele
1825 IF R.A<>INT(R.A) OR R.A<=0 OR R.A>R.N OR
    R.B<>INT(R.B) OR R.B<=0 OR R.B>R.N
    THEN ER%=-2
    ELSE R.AUX=V(R.A-1) :
    V(R.A-1)=V(R.B-1) :
    V(R.B-1)=R.AUX :
    ER%=0
1830 RETURN
5100 'Scrie un vector V() de S.N elemente
5105 IF INT(S.N)<>S.N OR S.N>20 OR S.N<=0
    THEN PRINT "ER : indice ";S.N;" eronat " :
    GOTO 5145
5110 IF ER%>0
    THEN PRINT "ER% =" ;ER% : GOTO 5145
5115 PRINT "Vectorul ";S.V$;" este :"
5120 FOR S.I=0 TO S.N-1 STEP 7
5125 IF S.I+6 <= S.N-1
    THEN S.F=S.I+6
    ELSE S.F=S.N-1
5130 FOR S.K=S.I TO S.F :
    PRINT USING " #####";V(S.K) ;
    NEXT
5135 PRINT

```



```

5140 NEXT
5145 RETURN
MERGE "C7"
Ok
RUN
Schimba doua elemente intre ele
N( 2 <= N <= 20 ) = 6
Vectorul este :
    10      6      7      3      6      8
Nr. element A ( 0 < Nr. <= 6 ) = 3
Nr. element B ( 0 < Nr. <= 6 ) = 5
Vectorul este :
    10      6      -6      3      7      8
Continuati ? (Y/N) : N
Ok

```

Listing 5.17

Rutina R16 interschimbă în vectorul V elementele de pe pozițiile R.A și R.B. Dacă R.A sau R.B nu aparțin intervalului $[0, R.N]$ se semnalează eroare (ER%=-2).

Rutina S4 permite afișarea cu format a conținutului vectorului V. De asemenea, prin variabila șir de caractere S.VȘ poate fi transmis și inserat un comentariu înaintea afișării propriu-zise a valorilor componentelor vectorului V.

● Inversarea ordinii elementelor unui vector

Aplicația, prin rutina R17, permite inversarea ordinii elementelor vectorului V de R.N elemente.

Schema logică a rutinei R17 este prezentată în figura 5.16.

Aplicația este prezentată în listingul 5.18. Elementele vectorului V sînt citite în linia 70 cu instrucțiunea READ din listele DATA (liniile 30 și 35). Rutina R17 conține liniile 1840 - 1850.

```

10 PRINT "Schimba ordinea elementelor lui V()"
20 DIM V(20)
30 DATA 10,6,7,3,-6,8,9,-11,6,24
35 DATA 4,11,-3,-8,0,-9,2,1,-1,21
40 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50 WHILE C$="Y" OR C$="Y"
60 GOSUB 3180 : N=C.A'Alege N
70 RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80 S.N=N : GOSUB 5100'Scrie vectorul
90 R.N=N : GOSUB 1840'Schimba ordine
100 GOSUB 140'Afișare
110 INPUT "Continuati ? (Y/N) : ",C$
120 WEND
130 END
140 'Afișare
150 IF ER%>0

```

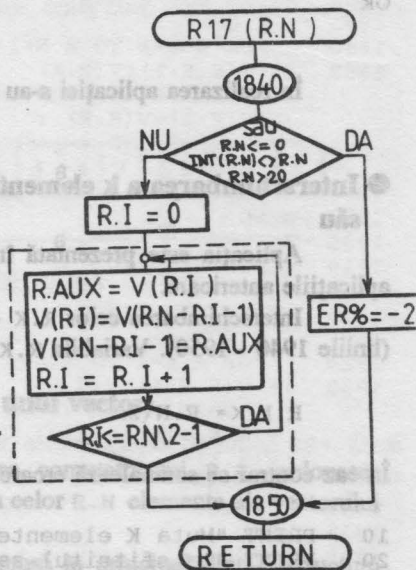


Fig. 5.16

5. Aplicații

```
THEN PRINT "ER =" ; ER%
ELSE S.N=N : GOSUB 5100
160 RETURN
1840 'Schimba ordinea elementelor lui V()
1845 IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
THEN ER%=-2
ELSE FOR R.I=0 TO R.N\2-1 :
R.AUX=V(R.I) :
V(R.I)=V(R.N-R.I-1) :
V(R.N-R.I-1)=R.AUX :
NEXT
1850 RETURN
MERGE "C7"
Ok
MERGE "S4"
Ok
RUN
Schimba ordinea elementelor lui V()
N( 2 <= N <= 20 ) = 6
Vectorul este :
10 6 7 3 -6 8
Vectorul este :
8 -6 3 7 6 10
Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 5
Vectorul este :
10 6 7 3 -6
Vectorul este :
-6 3 7 6 10
Continuati ? (Y/N) : N
Ok
```

Listing 5.18

În realizarea aplicației s-au utilizat de asemenea rutinele C7 și S4 (anexa A).

● Interschimbarea a k elemente de la începutul unui vector cu k elemente de la sfârșitul său

Aplicația este prezentată în listingul 5.19 și constituie o dezvoltare a celor prezentate în aplicațiile anterioare.

Interschimbarea celor R.K elemente se realizează cu rutina R19 prezentată în listingul 5.19 (liniile 1940 - 1950). Variabila R.K trebuie să îndeplinească condiția

$$R.K \leq R.N \setminus 2.$$

În caz contrar se semnalează eroare.

```
10 PRINT "Muta K elemente de la inceputul lui V()"
20 PRINT "la sfirsitul sau si invers"
30 DIM V(20)
40 DATA 10,6,7,3,-6,8,9,-11,6,24
35 DATA 4,11,-3,-8,0,-9,2,1,-1,21
```

```

50 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
60 WHILE C$="Y" OR C$="Y"
70   GOSUB 3180 : N=C.A'Alege N
80   RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
90   S.N=N : GOSUB 5100'Scrie vectorul.
100  PRINT "K ( 0 < K <= "N\2;" )";
110  INPUT " " = ",R.K
120  R.N=N : GOSUB 1940'Schimba
130  GOSUB 170'Afisare
140  INPUT "Continuati ? (Y/N) : ",C$
150  WEND
160  END
170  'Afisare
180  IF ER%<>0
      THEN PRINT "ER ="ER%
      ELSE S.N=N : GOSUB 5100
190  RETURN
1940 'Muta K elemente de la inceput la
1941 'sfirsitul lui V() si invers
1945 IF R.N<>INT(R.N) OR R.N<2 OR R.N>20
      THEN ER%=-5
      ELSE IF R.K<>INT(R.K) OR R.K<=0 OR R.K>R.N\2
            THEN ER%=-2
            ELSE FOR R.I=0 TO R.K-1:
R.AUX=V(R.I) :
V(R.I)=V(R.N-R.K+R.I) :
V(R.N-R.K+R.I)=R.AUX :
NEXT:ER%=0
1950 RETURN
MERGE "C7"
Ok
MERGE "S4"
Ok
RUN
Muta K elemente de la inceputul lui V()
la sfirsitul sau si invers
N( 2 <= N <= 20 ) = 6
Vectorul este :
      10      6      7      9      3      -6      8
K ( 0 < K <= 3 ) = 2
Vectorul este :
      -6      8      7      3      10      6
Continuati ? (Y/N) : N
Ok

```

Listing 5.19

● Ordonarea crescătoare / descrescătoare a elementelor unui vector

Aplicația permite prin rutina R18, funcție de valoarea comutatorului R.I, ordonarea crescătoare (dacă $R.I <> 0$) sau descrescătoare (dacă $R.I = 0$) a celor R.N elemente ale vectorului V.

Rutina R18, listingul 5.20, liniile 1860 - 1936, este compusă în principal dintr-o structură IF_THEN_ELSE care, funcție de valoarea comutatorului R.I dirijează execuția către rutina Descrescator ($R.I = 0$) sau Crescator ($R.I <> 0$). Ramurile fiind asemănătoare, în figura 5.17 se prezintă schema logică numai a rutinei Crescator (liniile 1910 - 1936).

5. Aplicații

```

10 PRINT "Ordonarea crescatoare descrescatoare a lui V()"
20 DIM V(20)
30 DATA 10,6,7,3,-6,8,9,-11,6,24
35 DATA 4,11,-3,-8,0,-9,2,1,-1,21
40 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50 WHILE C$="Y" OR C$="Y"
60   GOSUB 3180 : N=C.A'Alege N
70   RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80   S.N=N : GOSUB 5100'Scrie vectorul
90   INPUT "Crescator/Descrescator (*0) : ",R.I
100  R.N=N : GOSUB 1860'Ordonare
170  GOSUB 300'Afisare
180  INPUT "Continuati ? (Y/N) : ",C$
190 WEND
200 END
300 'Afisare
310 IF ER%<0
    THEN PRINT "ER =";ER%
    ELSE S.N=N : GOSUB 5100
320 RETURN
1860 'Ordonare crescatoare/descrescatoare
1861 IF R.N<INT(R.N) OR R.N<2 OR R.N>20
    THEN ER%=-5
    ELSE IF R.I=0
        THEN GOSUB 1870 : ER%=0
        ELSE GOSUB 1910 : ER%=0
1865 RETURN
1870 'Descrescator
1875 R.BUN=0
1880 WHILE NOT R.BUN
1885   R.BUN=-1
1890   FOR R.K=1 TO R.N-1
1895   IF V(R.K-1)<V(R.K)
        THEN R.AUX=V(R.K-1) :
        V(R.K-1)=V(R.K) :
        V(R.K)=R.AUX :
        R.BUN=0
1896   NEXT
1900 WEND
1905 RETURN
1910 'Crescator
1915 R.BUN=0
1920 WHILE NOT R.BUN
1921   R.BUN=-1
1925   FOR R.K=1 TO R.N-1
1930   IF V(R.K-1)>V(R.K)
        THEN R.AUX=V(R.K-1) :
        V(R.K-1)=V(R.K) :
        V(R.K)=R.AUX :
        R.BUN=0
1931   NEXT
1932 WEND
1935 RETURN
1936 WEND
MERGE "C7"
Ok
MERGE "S4"

```

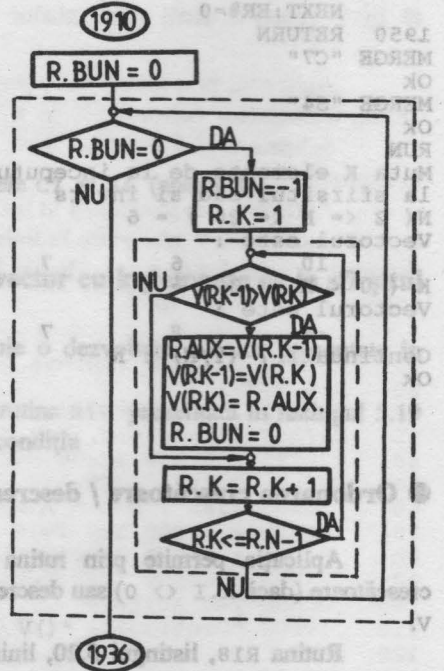


Fig. 5.17

```

Ok
RUN
Ordonarea crescatoare/descrescatoare a lui V()
N( 2 <= N <= 20 ) = 6
Vectorul este :
    10      6      7      3      -6      8
Crescator/descrescator (* / 0) : 0
Vectorul este :
    10      8      7      6      3      -6
Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 5
Vectorul este :
    10      6      7      3      -6
Crescator/descrescator (* / 0) : 1
Vectorul este :
    -6      3      6      7      10
Continuati ? (Y/N) : N
Ok

```

Listing 5.20

În realizarea aplicației s-au utilizat de asemenea rutinele C7 și S4 (anexa A).

● Rotirea stînga / dreapta a elementelor unui vector

Rotirea dreapta cu 3 poziții a elementelor unui vector V este prezentată schematic în figura 5.18. Rotirea s-a efectuat pas cu pas astfel încît elementul ajuns în extremitatea dreaptă este reținut pînă la deplasarea către dreapta cu o poziție a întregului vector, după care acesta trece pe prima poziție din stînga, și așa mai departe.

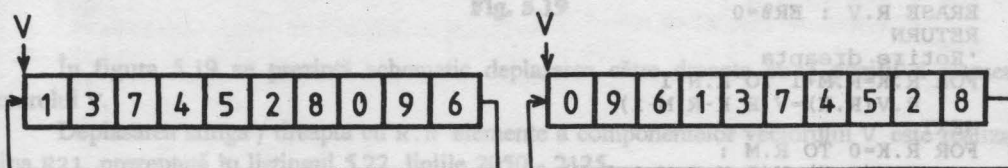


Fig. 5.18

Rotirea stînga / dreapta cu R.I elemente a vectorului V este efectuată de rutina R20. Dacă R.I < 0 se efectuează rotirea stînga a vectorului V cu R.I poziții, altfel rotirea se efectuează către dreapta. Rutina este prezentată în listingul 5.21, liniile 1960 - 2045.

```

10 PRINT "Rotirea stînga/dreapta a elementelor lui V()"
20 DIM V(20)
30 DATA 10,6,7,3,-6,8,9,-11,6,24
35 DATA 4,11,-3,-8,0,-9,2,1,-1,21

```

5. Aplicații

```

40 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50 WHILE C$="Y" OR C$="Y"
60   GOSUB 3180 : N=C.A'Alege N
70   RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80   S.N=N : GOSUB 5100'Scrie vectorul
90   INPUT "I = ",R.I
100  R.N=N : GOSUB 1960'Rotire
110  GOSUB 150'Afisare
120  INPUT "Continuati ? (Y/N) : ",C$
130 WEND
140 END
150 'Afisare
160 IF ER%<>0
    THEN PRINT "ER =" ;ER%
    ELSE S.N=N : GOSUB 5100
170 RETURN
1960 'Rotirea stinga/dreapta a lui V()
1965 IF R.N<>INT(R.N) OR R.N<2 OR R.N>20
    THEN ER%=-5
    ELSE IF R.I=0
        THEN ER%=0
        ELSE GOSUB 1975
1970 RETURN
1975 DIM R.V(20)
1980 R.J=ABS(R.I)
1985 IF R.J<R.N
    THEN R.M=R.J-1
    ELSE R.M=R.J MOD R.N
1990 IF R.I > 0
    THEN GOSUB 2010
    ELSE GOSUB 2030
1995 FOR R.K=0 TO R.N-1 :
    V(R.K)=R.V(R.K) :
NEXT
2000 ERASE R.V : ER%=0
2005 RETURN
2010 'Rotire dreapta
2015 FOR R.K=R.M+1 TO R.N-1 :
    R.V(R.K)=V(R.K-R.M-1) :
NEXT
2020 FOR R.K=0 TO R.M :
    R.V(R.K)=V(R.N-R.M-1+R.K) :
NEXT
2025 RETURN
2030 'Rotire stinga
2035 FOR R.K=R.M+1 TO R.N-1 :
    R.V(R.K-R.M-1)=V(R.K) :
NEXT
2040 FOR R.K=0 TO R.M :
    R.V(R.N-R.M-1+R.K)=V(R.K) :
NEXT
2045 RETURN
MERGE "C7"
OK
MERGE "S4"
OK
RUN
Rotirea stinga/dreapta a elementelor lui V()
N( 2 <= N <= 20 ) = 6

```

```

Vectorul este :
      10      6      7      3      -6      8
I = 2
Vectorul este :
      -6      8      10      6      7      3
Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 6
Vectorul este :
      10      6      7      3      -6      8
I = -2
Vectorul este :
      7      3      -6      8      10      6
Continuati ? (Y/N) : N
OK

```

Listing 5.21

În realizarea aplicației s-au mai utilizat și rutinele C7 și S4 (anexa A).

● Deplasarea stânga / dreapta a elementelor unui vector

Spre deosebire de rotire, deplasarea implică pierderea elementelor care datorită mutării în direcția dorită ies în afară; pozițiile rămase libere ca urmare a deplasării capătă automat valoarea 0.

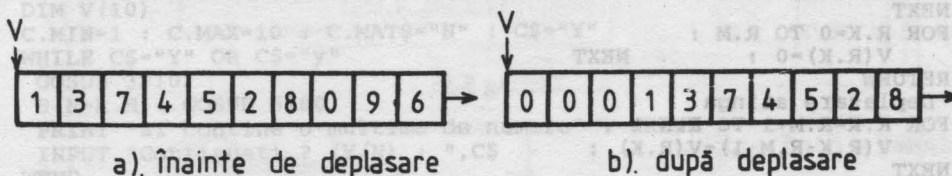


Fig. 5.19

În figura 5.19 se prezintă schematic deplasarea către dreapta cu 3 poziții a elementelor vectorului V.

Deplasarea stânga / dreapta cu R.N elemente a componentelor vectorului V este realizată de rutina R21 prezentată în listingul 5.22, liniile 2050 - 2125.

```

10 PRINT "Deplasarea stinga/dreapta a elementelor lui V()"
20 DIM V(20)
30 DATA 10,6,7,3,-6,8,9,-11,6,24
35 DATA 4,11,-3,-8,0,-9,2,1,-1,21
40 C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50 WHILE C$="Y" OR C$="Y"
60 GOSUB 3180 : N=C.A'Alege N
70 RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80 S.N=N : GOSUB 5100'Scrie vectorul
90 INPUT "I = ",R.I
100 R.N=N : GOSUB 2050'Deplasare
110 GOSUB 150'Afisare
120 INPUT "Continuati ? (Y/N) : ",C$
130 WEND
140 END

```

5. Aplicații

```

150 'Afisare
160 IF ER%<>0
    THEN PRINT "ER =" ; ER%
    ELSE S.N=N : GOSUB 5100
170 RETURN
2050 'Deplasare stinga/dreapta a lui V()
2055 IF R.N<>INT(R.N) OR R.N<2 OR R.N>20
    THEN ER%=-5
    ELSE IF R.I<>INT(R.I)
        THEN ER%=-2
        ELSE IF R.I=0
            THEN ER%=0
            ELSE GOSUB 2065

```

```

2060 RETURN
2065 R.J=ABS(R.I)
2070 IF R.J<R.N
    THEN R.M=R.J-1
    ELSE R.M=R.J MOD R.N
2075 IF R.I > 0
    THEN GOSUB 2090
    ELSE GOSUB 2110

```

```

2080 ER%=0
2085 RETURN
2090 'Deplasare dreapta
2095 FOR R.K=R.N-1 TO R.M+1 STEP -1 :
    V(R.K)=V(R.K-R.M-1) :
NEXT

```

```

2100 FOR R.K=0 TO R.M :
    V(R.K)=0 : NEXT
2105 RETURN
2110 'Deplasare stinga
2115 FOR R.K=R.M+1 TO R.N-1 :
    V(R.K-R.M-1)=V(R.K) :
NEXT

```

```

2120 FOR R.K=0 TO R.M :
    V(R.N-R.M-1+R.K)=0 :
NEXT
2125 RETURN
MERGE "C7"
Ok
MERGE "S4"
Ok
RUN

```

```

Deplasarea stinga/dreapta a elementelor lui V()
N( 2 <= N <= 20 ) = 6
Vectorul este :

```

```

I = 2
Vectorul este :

```

```

Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 6
Vectorul este :

```

```

I = -2
Vectorul este :

```

```

Continuati ? (Y/N) : N
Ok

```

Listing 5.22

5.3 Mulțimi

● Validarea unei mulțimi de numere

Într-o mulțime orice element este unic. În consecință validarea unei mulțimi de numere presupune verificarea unicității fiecărui element al său. Fie un vector V conținând $R.N$ valori numerice. Validarea se efectuează astfel:

- se ordonează crescător elementele acestuia;
- se verifică existența a cel puțin două elemente cu valori egale și:
 - dacă există atunci vectorul V nu conține o mulțime de numere;
 - dacă nu atunci vectorul V conține o mulțime de numere.

În cadrul aplicației prezentate în listingul 5.23, ordonarea elementelor vectorului V este efectuată utilizând rutina $R18$ (anexa A), iar verificarea unicității valorilor elementelor acestuia cu rutina $R22$ (listingul 5.23, liniile 2130 - 2170). În vederea simplificării construirii de aplicații care operează cu mulțimi de numere, rutinele $R18$ și $R22$ au fost cuplate împreună cu $C7$ în rutina $C8$ (listingul 5.23, liniile 3210 - 3260) care permite citirea, ordonarea crescătoare și validarea unei mulțimi de numere întregi.

```

10 PRINT "Citeste si verifica o multime de numere"
20 DIM V(10)
30 C.MIN=1 : C.MAX=10 : C.MATŞ="N" : CŞ="Y"
40 WHILE CŞ="Y" OR CŞ="y"
50 GOSUB 3210
60 S.N=R.N : GOSUB 5100
70 PRINT "si contine o multime de numere" : PRINT
80 INPUT "Continuati ? (Y/N) : " , CŞ
90 WEND
100 END
2130 'Verifica daca V() este o multime
2135 ER%=0 : R.I=1
2140 IF R.N<INT(R.N) OR R.N<=0 OR R.N>10
    THEN ER%=-11
    ELSE IF R.N => 2
        THEN GOSUB 2150
2145 RETURN
2150 WHILE R.I<R.N AND ER%=0
2155 IF V(R.I-1)=>V(R.I)
    THEN ER%=-10
    R.I=R.I+1
2165 WEND
2170 RETURN
3210 'Citeste si valideaza o multime V()
3215 ER%=-10
3220 WHILE ER%<>0
3225 GOSUB 3180 : R.N=C.A'Citeste N
3230 PRINT "Introduceti multimea de";R.N;"numere : "
3235 FOR R.I=1 TO R.N :
PRINT " N";:PRINT USING "$#";R.I;:
INPUT,"= ",V(R.I-1);
NEXT : PRINT
3240 R.I=1 : GOSUB 1860'Ordonare

```

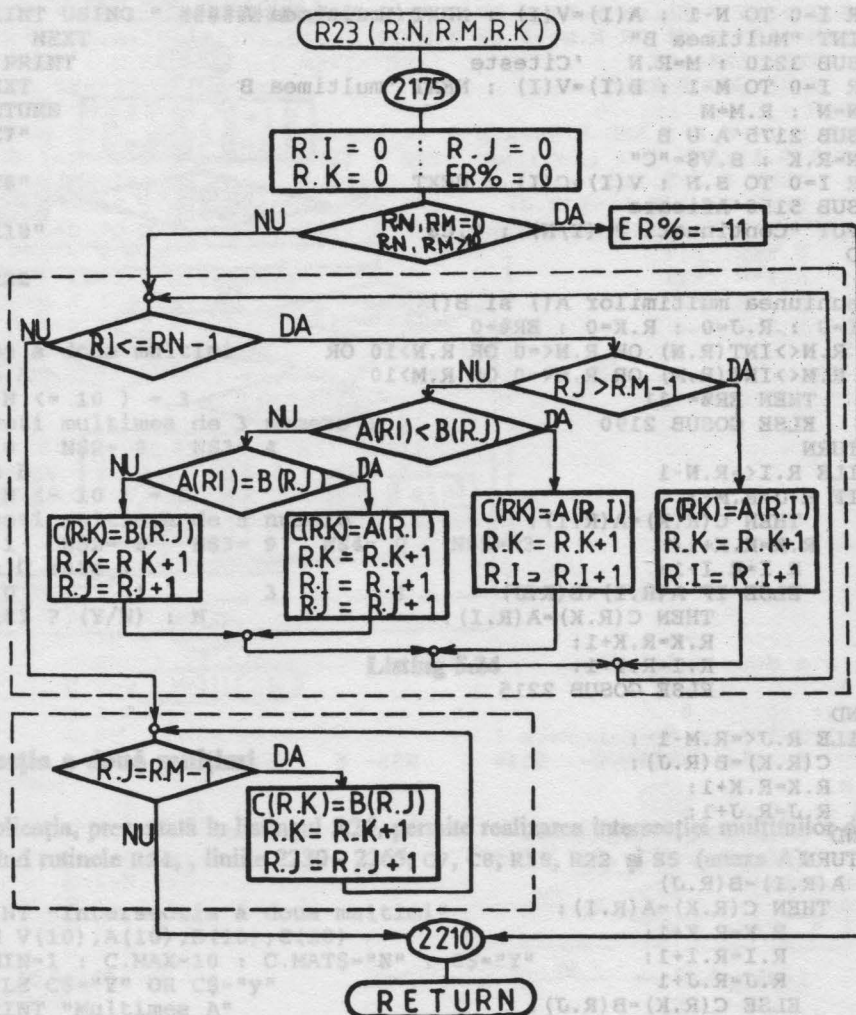



Fig. 5.20

Aplicația, prezentată în listingul 5.24, utilizează pe lângă rutina R23 (liniile 2175 - 2220) și S5 (liniile 5150 - 5195), rutinele C7, C8, R18 și R22 (anexa A).

```

10 PRINT "Reuniunea a doua multimi"
20 DIM V(10),A(10),B(10),C(20)
30 C.MIN=1 : C.MAX=10 : C.MATŞ="N" : CŞ="Y"
40 WHILE CŞ="Y" OR CŞ="y"
50 PRINT "Multimea A"
60 GOSUB 3210 : N=R.N 'Citeste
  
```

5. Aplicații

```

70   FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80   PRINT "Multimea B"
90   GOSUB 3210 : M=R.N 'Citeste
100  FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110  R.N=N : R.M=M
120  GOSUB 2175 'A U B
130  S.N=R.K : S.V$="C"
140  FOR I=0 TO S.N : V(I)=C(I) : NEXT
150  GOSUB 5150 'Afisare
160  INPUT "Continuati ? (Y/N) : ", C$
170  WEND
180  END

```

```

2175 'Reuniunea multimilor A() si B()
2180 R.I=0 : R.J=0 : R.K=0 : ER%=0
2185 IF R.N<>INT(R.N) OR R.N<=0 OR R.N>10 OR
    R.M<>INT(R.M) OR R.M<=0 OR R.M>10
    THEN ER%=-11
    ELSE GOSUB 2190

```

```
2186 RETURN
```

```
2190 WHILE R.I<=R.N-1
```

```
2195 IF R.J>R.M-1
```

```
    THEN C(R.K)=A(R.I) :
```

```
    R.K=R.K+1 :
```

```
    R.I=R.I+1 :
```

```
    ELSE IF A(R.I)<B(R.J)
```

```
        THEN C(R.K)=A(R.I) :
```

```
        R.K=R.K+1 :
```

```
        R.I=R.I+1 :
```

```
        ELSE GOSUB 2215
```

```
2200 WEND
```

```
2205 WHILE R.J<=R.M-1 :
```

```
    C(R.K)=B(R.J) :
```

```
    R.K=R.K+1 :
```

```
    R.J=R.J+1 :
```

```
WEND
```

```
2210 RETURN
```

```
2215 IF A(R.I)=B(R.J)
```

```
    THEN C(R.K)=A(R.I) :
```

```
    R.K=R.K+1 :
```

```
    R.I=R.I+1 :
```

```
    R.J=R.J+1
```

```
    ELSE C(R.K)=B(R.J) :
```

```
    R.K=R.K+1 :
```

```
    R.J=R.J+1
```

```
2220 RETURN
```

```
5150 'Scrie un vector V() de S.N elemente
```

```
5155 IF INT(S.N)<>S.N OR S.N>20 OR S.N<=0
```

```
    THEN PRINT "ER : indice ";S.N;" eronat " :
```

```
    GOTO 5195
```

```
5160 IF ER%>0
```

```
    THEN PRINT "ER% =";ER% : GOTO 5195
```

```
5165 PRINT "Multimea ";S.V$;" este " :
```

```
5170 FOR S.I=0 TO S.N-1 STEP 7
```

```
5175 IF S.I+6 <= S.N-1
```

```
    THEN S.F=S.I+6
```

```
    ELSE S.F=S.N-1
```

```
5180 FOR S.K=S.I TO S.F :
```

```

10 PRINT USING "#####";V(S.K);M;
20 NEXT
5185 PRINT
5190 NEXT
5195 RETURN
MERGE "C7"
Ok
MERGE "C8"
Ok
MERGE "R18"
Ok
MERGE "R22"
Ok
RUN
Reuniunea a doua multimi
Multimea A
N( 1 <= N <= 10 ) = 3
Introduceti multimea de 3 numere :
  N$1= 0  N$2= 8  N$3= 4
Multimea B
N( 1 <= N <= 10 ) = 5
Introduceti multimea de 5 numere :
  N$1= 1  N$2= 8  N$3= 9  N$4= 0  N$5= 3
Multimea C este :
      0      1      3      4      8      9
Continuati ? (Y/N) : N
Ok

```

Listing 5.24

● Intersecția a două mulțimi

Aplicația, prezentată în listingul 5.25, permite realizarea intersecției mulțimilor de numere A și B utilizând rutinele R24,, liniile 2230 - 2265, C7, C8, R18, R22 și S5 (anexa A).

```

10 PRINT "Intersecția a doua multimi"
20 DIM V(10),A(10),B(10),C(20)
30 C.MIN=1 : C.MAX=10 : C.MAT$="N" : C$="Y"
40 WHILE C$="Y" OR C$="y"
50 PRINT "Multimea A"
60 GOSUB 3210 : N=R.N 'Citeste
70 FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80 PRINT "Multimea B"
90 GOSUB 3210 : M=R.N 'Citeste
100 FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110 R.N=N : R.M=M
120 GOSUB 2230 'Intersecție
130 S.N=R.K : S.V$="C"
140 FOR I=0 TO S.N : V(I)=C(I) : NEXT
150 GOSUB 5150 'Afișare
160 INPUT "Continuati ? (Y/N) : ",C$
170 WEND
180 END
2230 'Intersecția multimilor A() și B()
2235 R.I=0 : R.J=0 : R.K=0 : ER%=0

```

5. Aplicații

```

2240 IF R.N<>INT(R.N) OR R.N<=0 OR R.N>10 OR
      R.M<>INT(R.M) OR R.M<=0 OR R.M>10
      THEN ER%=-11
      ELSE GOSUB 2250
2245 RETURN
2250 WHILE R.I<=R.N-1 AND R.J<=R.M-1
2255 IF A(R.I) < B(R.J)
      THEN R.I=R.I+1
      ELSE IF A(R.I)=B(R.J)
          THEN C(R.K)=A(R.I):
               R.K=R.K+1:
               R.I=R.I+1:
               R.J=R.J+1
          ELSE R.J=R.J+1
2260 WEND
2265 RETURN
MERGE "C7"
Ok
MERGE "C8"
Ok
MERGE "R18"
Ok
MERGE "R22"
Ok
MERGE "S5"
Ok
RUN

```

```

Intersectia a doua multimi
Multimea A
N( 1 <= N <= 10 ) = 6
Introduceti multimea de 6 numere :
  N$1= 1  N$2= 4  N$3= 0  N$4= 8  N$5= 6  N$6= 2
Multimea B
N( 1 <= N <= 10 ) = 4
Introduceti multimea de 4 numere :
  N$1= 3  N$2= 8  N$3= 0  N$4= 7
Multimea C este :
      0      8
Continuati ? (Y/N) : N
Ok

```

Listing 5.25

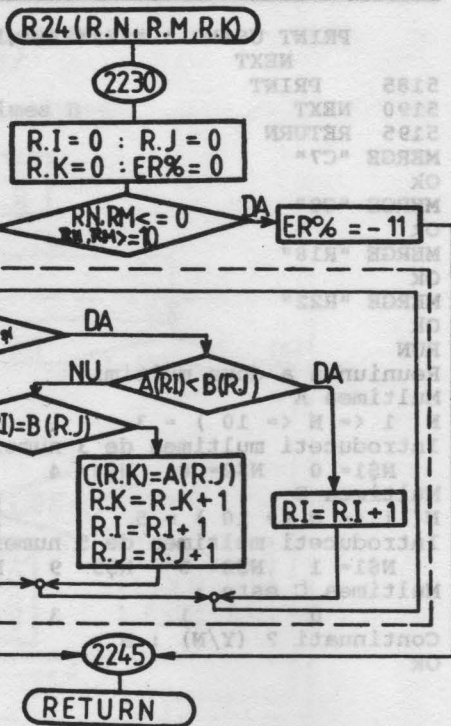


Fig. 5.21

Rutina R24 realizează intersecția mulțimilor ordonate A și B avînd R.N respectiv R.M elemente. Rezultatul este deus în vectorul C iar numărul elementelor acesteia este atribuit variabilei R.K. Schema logică a rutinei R24 este prezentată în figura 5.21.

● Diferența a două mulțimi

Fie A și B două mulțimi de R.N respectiv R.M elemente. Mulțimea $C = A \setminus B$ va conține toate elementele care aparțin de A și nu aparțin de B.

Diferența mulțimilor A și B este realizată de rutina R25 a cărei schemă logică este prezentată în figura 5.22. Textul sursă al rutinei este prezentat în listingul 5.26, liniile 2280 - 2325.

```

10 PRINT "Diferenta a doua multimi"
20 DIM V(10),A(10),B(10),C(20)
30 C.MIN=1 : C.MAX=10 : C.MAT$="N" : C$="Y"
40 WHILE C$="Y" OR C$="y"
50 PRINT "Multimea A"
60 GOSUB 3210 : N=R.N 'Citeste
70 FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80 PRINT "Multimea B"
90 GOSUB 3210 : M=R.N 'Citeste
100 FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110 R.N=N : R.M=M
120 GOSUB 2280 'Diferenta
130 S.N=R.K : S.V$="C"
140 FOR I=0 TO S.N : V(I)=C(I) : NEXT
150 GOSUB 5150 'Afisare
160 INPUT "Continuati ? (Y/N) : ",C$
170 WEND
180 END
2280 'Diferenta multimilor A() \ B()
2285 R.I=0 : R.J=0 : R.K=0 : ER%=0
2290 IF R.N<>INT(R.N) OR R.N<=0 OR R.N>10 OR
    R.M<>INT(R.M) OR R.M<=0 OR R.M>10
    THEN ER%=-11
    ELSE GOSUB 2300
2295 RETURN
2300 WHILE R.I<=R.N-1
2305 IF R.J>R.M
    THENC(R.K)=A(R.I) :
    R.K=R.K+1 :
    R.I=R.I+1
    ELSE IF A(R.I)=B(R.J)
    THEN R.I=R.I+1 :
    R.J=R.J+1
    ELSE R.J=R.J+1
2325 RETURN
MERGE "C7"
Ok
MERGE "C8"
Ok
MERGE "R18"
Ok
MERGE "R22"
Ok
MERGE "S5"
Ok
RUN
Diferenta a doua multimi
Multimea A
N ( 1 <= N <= 10 ) = 6
Introduceti multimea de 6 numere :

```

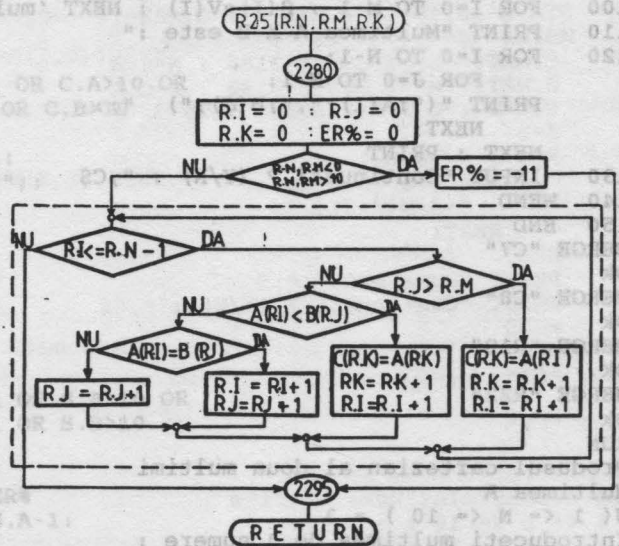


Fig. 5.22

5. Aplicații

```

NŞ1= 1   NŞ2= 2   NŞ3= 5   NŞ4= 7   NŞ5= 9   NŞ6= 8
Multimea B
N( 1 <= N <= 10 ) = 3
Introduceti multimea de 3 numere :
NŞ1= 5   NŞ2= 0   NŞ3= 8
Multimea C este :
1         2
Continuati ? (Y/N) : N
Ok

```

Listing 5.26

● Produsul cartezian al două mulțimi de numere

Aplicația permite citirea, ordonarea și validarea mulțimilor A și B de N respectiv M elemente și determină produsul cartezian $A \times B$ fără a reține însă mulțimea perechilor formate.

Aplicația este prezentată în listingul 5.27.

```

10 PRINT "Produsul cartezian al doua multimi"
20 DIM V(10),A(10),B(10)
30 C.MIN=1 : C.MAX=10 : C.MATŞ="N" : CŞ="Y"
40 WHILE CŞ="Y" OR CŞ="y"
50 PRINT "Multimea A"
60 GOSUB 3210 : N=R.N 'Citeste
70 FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80 PRINT "Multimea B"
90 GOSUB 3210 : M=R.N 'Citeste
100 FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110 PRINT "Multimea A X B este : "
120 FOR I=0 TO N-1 :
    FOR J=0 TO M-1 :
        PRINT "(";A(I);";";B(J);") " ; ;
    NEXT J
NEXT I : PRINT
130 INPUT "Continuati ? (Y/N) : " , CŞ
140 WEND
150 END

```

MERGE "C7"

Ok

MERGE "C8"

Ok

MERGE "R18"

Ok

MERGE "R22"

Ok

RUN

Produsul cartezian al doua multimi

Multimea A

N(1 <= N <= 10) = 3

Introduceti multimea de 3 numere :

NŞ1= 1 NŞ2= 4 NŞ3= 6

Multimea B

N(1 <= N <= 10) = 2

Introduceti multimea de 2 numere :


```

NȘ1= 1   NȘ2= 5
Multimea A X B este :
( 1 , 1 ) ( 1 , 5 ) ( 4 , 1 ) ( 4 , 5 ) ( 6 , 1 ) ( 6 , 5 )
Continuati ? (Y/N) : N
Ok

```

Listing 5.27

5.4 Matrici, determinanți

● Citirea și listarea unei matrice

Aplicația oferă o variantă de rutine pentru citirea și validarea elementelor unei matrici de numere întregi (rutina C9) precum și pentru listarea acesteia utilizând rutina S6. Prezentată în listingul 5.28 aplicația conține alături de programul principal (liniile 10 - 110), rutinele C9 (liniile 3270 - 3280) și S6 (liniile 5200 - 5210).

```

10 PRINT "Citeste si listeaza o matrice"
20 DIM M(10,10) : C.MIN=1 : C.MAX=10 : CȘ="Y"
30 WHILE CȘ="Y" OR CȘ="y"
40 C.MATȘ="N" : GOSUB 3180 : N=C.A
50 C.MATȘ="M" : GOSUB 3180 : M=C.A
60 C.A=N : C.B=M : GOSUB 3270
70 PRINT "Matricea M este:"
80 S.A=N : S.B=M : GOSUB 5250
90 INPUT "Continuati ? (Y/N) : ",CȘ
100 WEND
110 END

3270 'Citeste matricea M()
3275 IF C.A<>INT(C.A) OR C.A<1 OR C.A>10 OR
    C.B<>INT(C.B) OR C.B<1 OR C.B>10
    THEN ER%=-12
    ELSE FOR C.I=0 TO C.A-1:
        PRINT "Linia";C.I+1;";";:
        FOR C.J=0 TO C.B-1:
            INPUT;" ",M(C.I,C.J):
            PRINT ",,":
        NEXT:
        PRINT:
    NEXT
    RETURN

5200 'Afiseaza matricea M()
5205 IF S.A<>INT(S.A) OR S.A<1 OR S.A>10 OR
    S.B<>INT(S.B) OR S.B<1 OR S.B>10
    THEN PRINT "ER = -12"
    ELSE IF ER%<>0
        THEN PRINT "ER =",ER%
        ELSE FOR S.I=0 TO S.A-1:
            FOR S.J=0 TO S.B-1:
                PRINT M(S.I,S.J);:
            NEXT:PRINT:
        NEXT
    RETURN

```

```

MERGE "C7"
Ok
RUN
Citeste si listeaza o matrice
N( 1 <- N <- 10 ) = 3
M( 1 <- N <- 10 ) = 5
Linia 1 : 1, 4, 6, 7, 8,
Linia 2 : 9, 0, 0, 8, 9,
Linia 3 : 1, 3, 4, 6, 7,
Matricea M este :
  1 4 6 7 8
  9 0 0 8 9
  1 3 4 6 7
Continuati ? (Y/N) : N
Ok
    
```

Listing 5.28

Pentru citirea și validarea variabilelor N (numărul de linii) și M (numărul de coloane) care definesc dimensiunile matricei M este utilizată rutina C7.

● Suma elementelor unei matrice

În realizarea acestei aplicații, prezentată în listingul 5.29, s-au utilizat rutinele C7 (pentru citirea și validarea unui număr întreg cuprins între C.MIN și C.MAX) și C9 (citirea și validarea elementelor unei matrici de C.A X C.B componente).

```

10 PRINT "Suma elementelor unei matrice"
20 DIM M(10,10) : C.MIN=1 : C.MAX=10 : C$="Y"
30 WHILE C$="Y" OR C$="y"
40 C.MAT$="N" : GOSUB 3180 : N=C.A
50 C.MAT$="M" : GOSUB 3180 : M=C.A
60 C.A=N : C.B=M : GOSUB 3270
70 S=0
80 FOR I=0 TO N-1:
  FOR J=0 TO M-1:
    S=S+M(I,J):
  NEXT:
  NEXT
90 PRINT "Suma este :";S
100 INPUT "Continuati ? (Y/N) : ",C$
110 WEND
120 END
MERGE "C7"
Ok
MERGE "C9"
Ok
RUN
Suma elementelor unei matrice
N( 1 <- N <- 10 ) = 5
M( 1 <- N <- 10 ) = 6
Linia 1 : 1, 4, 8, 9, 0, 2,
Linia 2 : 3, 2, 1, 7, 0, 6,
Linia 3 : 5, 8, 9, 2, 4, 3,
    
```

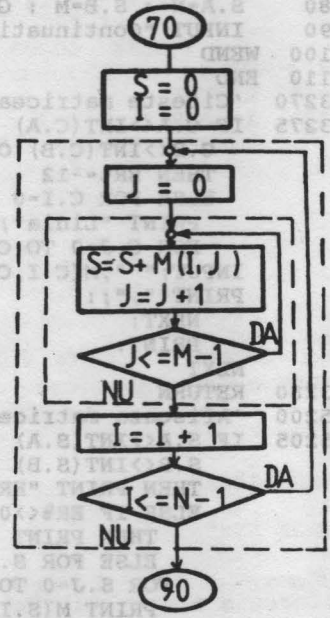


Fig. 5.23

Linia 4 : 7, 8, 9, 1, 3, 4,

Linia 5 : 4, 6, 7, 7, 2, 9,

Suma este : 141

Continuati ? (Y/N) : N

Ok

Listing 5.29

Suma propriu-zisă a elementelor matricii M este realizată în secvența de program cuprinsă între liniile 70 la 90. Schema logică a secvenței este prezentată în figura 5.23. Variabilele N și M reprezintă dimensiunile matricii M iar variabila S conține suma acestora.

● Suma a două matrici

Aplicația, prezentată în listingul 5.30, permite citirea și validarea elementelor matricilor A și B de $N \times M$ componente (liniile 50 - 120), determinarea sumei acestora (linia 130) precum și afișarea matricii sumă M (liniile 140 - 150).

```
10 PRINT "Suma a doua matrici (n X m)"
```

```
20 DIM M(10,10),A(10,10),B(10,10)
```

```
30 C.MIN=1 : C.MAX=10 : C$="Y"
```

```
40 WHILE C$="Y" OR C$="y"
```

```
50 C.MAT$="N" : GOSUB 3180 : N=C.A
```

```
60 C.MAT$="M" : GOSUB 3180 : M=C.A
```

```
70 PRINT "Matricea A :"
```

```
80 C.A=N : C.B=M : GOSUB 3270
```

```
90 FOR I=0 TO N-1 :
```

```
FOR J=0 TO M-1:A(I,J)=M(I,J):NEXT
```

```
NEXT
```

```
100 PRINT "Matricea B :"
```

```
110 GOSUB 3270
```

```
120 FOR I=0 TO N-1 :
```

```
FOR J=0 TO M-1:B(I,J)=M(I,J):NEXT
```

```
NEXT
```

```
130 FOR I=0 TO N-1 :
```

```
FOR J=0 TO M-1:
```

```
M(I,J)=A(I,J)+B(I,J):
```

```
NEXT:
```

```
NEXT
```

```
140 PRINT "Matricea M = A + B este :"
```

```
150 S.A=N : S.B=M : GOSUB 5200
```

```
160 INPUT "Continuati ? (Y/N) : ",C$
```

```
170 WEND
```

```
180 END
```

```
MERGE "C7"
```

Ok

```
MERGE "C9"
```

Ok

```
MERGE "S6"
```

Ok

```
RUN
```

Suma a doua matrici (n X m)

5. Aplicații

$N(1 \leq N \leq 10) = 3$

$M(1 \leq N \leq 10) = 4$

Matricea A :

Linia 1 : 8, 9, 6, 4,

Linia 2 : 2, 6, 8, 5,

Linia 3 : 5, 7, 2, 1,

Matricea B :

Linia 1 : 9, 7, 7, 8,

Linia 2 : 8, 4, 3, 9,

Linia 3 : 7, 9, 8, 9,

Matricea $M = A + B$ este :

17 16 13 12

10 10 11 14

12 16 10 10

Continuati ? (Y/N) : N

Ok

Listing 5.30

Observație. Deoarece citirea și validarea elementelor matricilor A și B s-a efectuat cu rutina C9 (anexa A) care depune valorile elementelor citite în matricea M de C.A X C.B elemente, după revenirea din rutină este necesară transferarea elementelor matricei M în matricea A (linia 90) respectiv B (linia 120).

● Produsul a două matrici

Aplicația (listingul 5.31) permite determinarea produsului matricilor A și B de $(N \times M)$ respectiv $(M \times P)$ elemente. Rezultatul este stocat în matricea M de $(N \times P)$ elemente.

Citirea și validarea elementelor matricilor A și B se efectuează utilizând rutinele C7 și C9 (anexa A). Produsul efectiv al matricilor A și B se realizează în secvența din linia 150 (listingul 5.31).

```
10 PRINT "Produsul a doua matrici"
20 PRINT "A(n X m) si B(m X p)"
30 DIM M(10,10),A(10,10),B(10,10)
40 C.MIN=1 : C.MAX=10 : C$="Y"
50 WHILE C$="Y" OR C$="y"
60 C.MAT$="N" : GOSUB 3180 : N=C.A
70 C.MAT$="M" : GOSUB 3180 : M=C.A
80 C.MAT$="P" : GOSUB 3180 : P=C.A
90 PRINT "Matricea A(";N;"X";M;") : "
100 C.A=N : C.B=M : GOSUB 3270
110 FOR I=0 TO N-1 :
    FOR J=0 TO M-1:A(I,J)=M(I,J):NEXT:
    NEXT
120 PRINT "Matricea B(";M;"X";P;") : "
130 C.A=M : C.B=P : GOSUB 3270
140 FOR I=0 TO M-1 :
    FOR J=0 TO P-1:B(I,J)=M(I,J):NEXT:
    NEXT
150 FOR I=0 TO N-1 :
    FOR J=0 TO P-1 :
        M(I,J)=0:
```

```

FOR K=0 TO M-1:
M(I,J)=M(I,J)+A(I,K)*B(K,J):
NEXT:

```

```

NEXT:

```

```

160 PRINT "Matricea M = A X B este : "
170 S.A=N : S.B=P : GOSUB 5200
180 INPUT "Continuati ? (Y/N) : ",CŞ
190 WEND
200 END

```

```

MERGE "C7"

```

```

Ok

```

```

MERGE "C9"

```

```

Ok

```

```

MERGE "S6"

```

```

Ok

```

```

RUN

```

Produsul a doua matrici

A(n X m) si B(m X p)

N(1 <= N <= 10) = 4

M(1 <= N <= 10) = 3

P(1 <= P <= 10) = 5

Matricea A(4 X 3) :

Linia 1 : 1, 4, 3,

Linia 2 : 2, 1, 5,

Linia 3 : 3, 2, 3,

Linia 4 : 2, 1, 4,

Matricea B(3 X 5) :

Linia 1 : 2, 5, 7, 6, 0,

Linia 2 : 3, 1, 4, 5, 2,

Linia 3 : 2, 8, 1, 3, 3,

Matricea M = A X B este :

20	33	26	35	17
17	51	23	32	17
18	41	32	37	13
15	43	22	29	14

Continuati ? (Y/N) : N

Ok

Listing 5.31

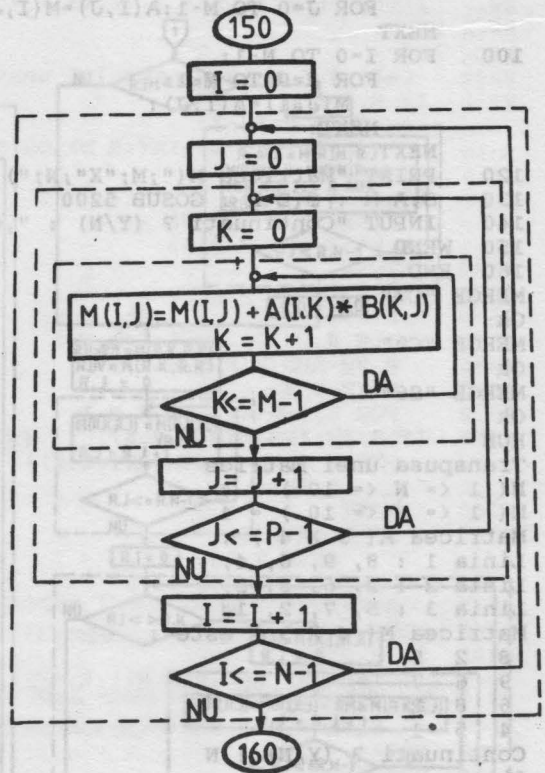


Fig. 5.24

Schema logică a acestei secvențe este prezentată în figura 5.24.

● Transpusa unei matrice

Aplicația este prezentată în listingul 5.32. Pentru determinarea transpusei matricei M de (N X M) componente s-a utilizat matricea auxiliară A. În realizarea aplicației s-au utilizat de asemenea rutinele C7, C9 și S6 (anexa A).

```

10 PRINT "Transpusa unei matrice"
20 DIM M(10,10),A(10,10)
30 C.MIN=1 : C.MAX=10 : CŞ="Y"
40 WHILE CŞ="Y" OR CŞ="y"
50 C.MATŞ="N" : GOSUB 3180 : N=C.A
60 C.MATŞ="M" : GOSUB 3180 : M=C.A
70 PRINT "Matricea A(";N;"X";M;") : "

```

5. Aplicații

```

80 C.A=N : C.B=M : GOSUB 3270
90 FOR I=0 TO N-1 :
    FOR J=0 TO M-1:A(I,J)=M(I,J):NEXT J
    NEXT I
100 FOR I=0 TO N-1:
    FOR J=0 TO M-1:
        M(J,I)=A(I,J):
    NEXT J
    NEXT I
120 PRINT "Matricea M(";M;"X";N;") este : "
130 S.A=M : S.B=N : GOSUB 5200
140 INPUT "Continuati ? (Y/N) : ",C$
150 WEND
160 END
MERGE "C7"
Ok
MERGE "C9"
Ok
MERGE "S6"
Ok
Ok
RUN
Transpusa unei matrice
N ( 1 <= N <= 10 ) = 3
M ( 1 <= N <= 10 ) = 4
Matricea A ( 3 X 4 ) :
Linia 1 : 8, 9, 6, 4,
Linia 2 : 2, 6, 8, 5,
Linia 3 : 5, 7, 2, 1,
Matricea M ( 4 X 3 ) este :
8 2 5
9 6 7
6 8 2
4 5 *1
Continuati ? (Y/N) : N
Ok

```

Listing 5.32

● Calculul determinantului unei matrice

Aplicația permite determinarea cu o aproximație de $1/10000$ a determinantului unei matrice pătrate de ordin N ($0 < N \leq 10$). Calculul valorii determinantului matricei M este realizat de rutina R26 a cărei schemă logică este prezentată în figura 5.25.

Variabilele de intrare în subrutină sînt R.EPS (conținînd valoarea aleasă pentru eroarea admisă) și R.N (ordinul matricei). Matricea M de rang $R.N$ se consideră variabila globală. Valoarea determinantului este depusă în variabila de ieșire R.D. În cazul cînd valoarea determinantului matricei M este mai mică decît R.EPS aceasta se consideră 0.

Aplicația este prezentată în listingul 5.33. Liniile 2330 - 2450 aparțin matricei R26.

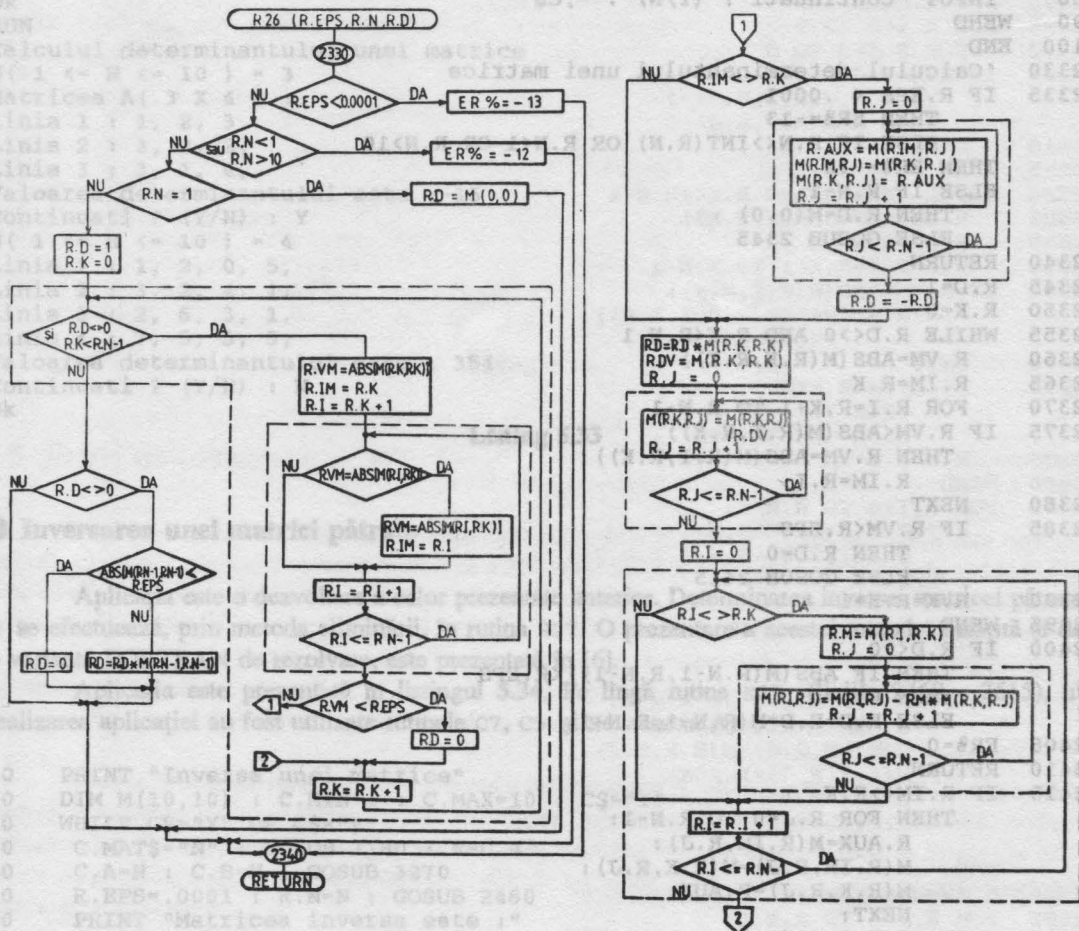


Fig. 5.25

```

10 PRINT "Calculul determinantului unei matrice"
20 DIM M(10,10) : C.MIN=1 : C.MAX=10 : C$="Y"
30 WHILE C$="Y" OR C$="y"
40   C.MAT$="N" : GOSUB 3180 : N=C.A
50   C.A=N : C.B=N : GOSUB 3270
60   R.EPS=.0001 : R.N=N : GOSUB 2330
70   IF ER%=0
      THEN PRINT "Valoarea determinantului este :";

```

5. Aplicații

R.D

ELSE PRINT "ER =", ER%

80 INPUT "Continuati ? (Y/N) : ", C\$

90 WEND

100 END

2330 'Calculul determinantului unei matrice

2335 IF R.EPS < .0001

THEN ER%=-13

ELSE IF R.N<>INT(R.N) OR R.N<1 OR R.N>10

THEN ER%=-12

ELSE IF R.N=1

THEN R.D=M(0,0)

ELSE GOSUB 2345

2340 RETURN

2345 R.D=1

2350 R.K=0

2355 WHILE R.D<>0 AND R.K<R.N-1

R.VM=ABS(M(R.K,R.K))

R.IM=R.K

2370 FOR R.I=R.K+1 TO R.N-1

2375 IF R.VM<ABS(M(R.I,R.K))

THEN R.VM=ABS(M(R.I,R.K)) :

R.IM=R.I

2380 NEXT

2385 IF R.VM<R.EPS

THEN R.D=0

ELSE GOSUB 2415

2390 R.K=R.K+1

2395 WEND

2400 IF R.D<>0

THEN IF ABS(M(R.N-1,R.N-1))<R.EPS

THEN R.D=0

ELSE R.D=R.D*M(R.N-1,R.N-1)

2405 ER%=0

2410 RETURN

2415 IF R.IM<>R.K

THEN FOR R.J=0 TO R.N-1:

R.AUX=M(R.IM,R.J) :

M(R.IM,R.J)=M(R.K,R.J) :

M(R.K,R.J)=R.AUX :

NEXT :

R.D=R.D*(-1)

2420 R.D=R.D*M(R.K,R.K)

2425 R.DV=M(R.K,R.K)

2430 FOR R.J=0 TO R.N-1:

M(R.K,R.J)=M(R.K,R.J)/R.DV:

NEXT

2435 FOR R.I=0 TO R.N-1

2440 IF R.I<>R.K

THEN R.M=M(R.I,R.K) :

FOR R.J=0 TO R.N-1:

M(R.I,R.J)=M(R.I,R.J)-R.M*

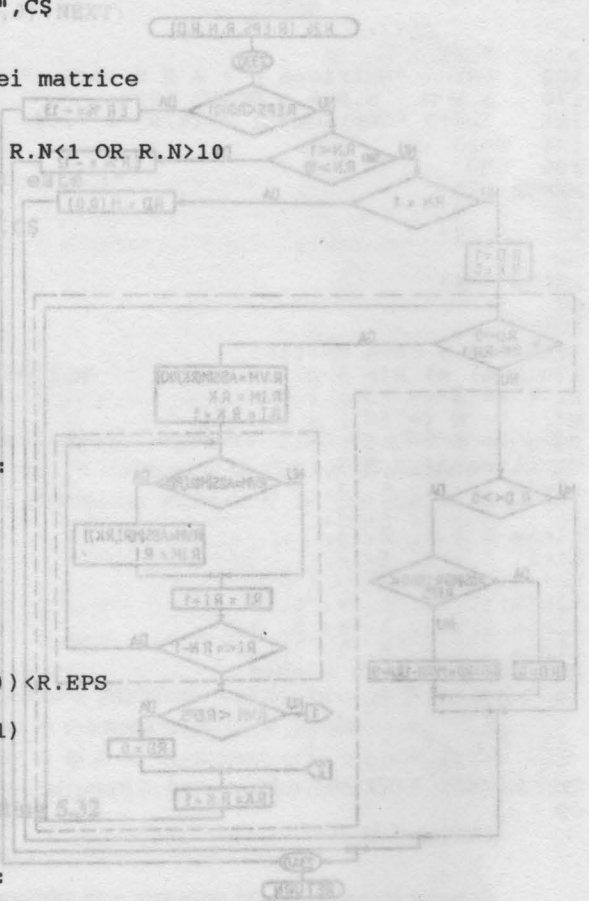
M(R.K,R.J) :

NEXT

2445 NEXT

2450 RETURN

MERGE "C7"




```

Ok
MERGE "C9"
Ok
RUN
Calculul determinantului unei matrice
N( 1 <= N <= 10 ) = 3
Matricea A( 3 X 4 ) :
Linia 1 : 1, 2, 3,
Linia 2 : 3, 2, 2,
Linia 3 : 3, 1, 4,
Valoarea determinantului este :-15
Continuati ? (Y/N) : Y
N( 1 <= N <= 10 ) = 4
Linia 1 : 1, 2, 0, 5,
Linia 2 : 3, 2, 4, 1,
Linia 3 : 2, 6, 3, 1,
Linia 4 : 7, 5, 3, 5,
Valoarea determinantului este : 354
Continuati ? (Y/N) : N
Ok

```

Listing 5.33

● Inversarea unei matrici pătrate

Aplicația este o dezvoltare a celor prezentate anterior. Determinarea inversei matricii pătrate M se efectuează, prin metoda eliminării, în rutina R27. O prezentare a acestei metode, însoțită și de o variantă FORTRAN de rezolvare, este prezentată în [6].

Aplicația este prezentată în listingul 5.34. Pe lângă rutina R27, (liniile 2460 - 2615), în realizarea aplicației au fost utilizate rutinele C7, C5 și S6 (anexa A).

```

10 PRINT "Inversa unei matrice"
20 DIM M(10,10) : C.MIN=1 : C.MAX=10 : C$="Y"
30 WHILE C$="Y" OR C$="y"
40 C.MAT$="N" : GOSUB 3180 : N=C.A
50 C.A=N : C.B=N : GOSUB 3270
60 R.EPS=.0001 : R.N=N : GOSUB 2460
70 PRINT "Matricea inversa este :"
80 S.A=N : S.B=N : GOSUB 5200
90 INPUT "Continuati ? (Y/N) : ", C$
100 WEND
110 END
2460 'Inversarea unei matrice
2465 IF R.EPS < .0001
    THEN ER%= -13
    ELSE IF R.N <> INT(R.N) OR R.N < 1 OR R.N > 10
        THEN ER%= -12
        ELSE IF R.N = 1
            THEN IF ABS(M(0,0)) < R.EPS
                THEN ER%= -14
                ELSE M(0,0) = 1/M(0,0)
            ELSE GOSUB 2475
2470 RETURN
2475 ER%= 0

```

5. Aplicații

```

2480 R.D=1
2485 R.K=0
2490 DIM R.M(10,10)
2495 FOR R.I=0 TO 9
2500   FOR R.J=0 TO 9
2505   IF R.I=R.J
      THEN R.M(R.I,R.J)=1
      ELSE R.M(R.I,R.J)=0
2510   NEXT
2515 NEXT
2520 WHILE ER%=0 AND R.K<R.N-1
2525   R.VM=ABS(M(R.K,R.K))
2530   R.IM=R.K
2535   FOR R.I=R.K+1 TO R.N-1
2540   IF R.VM<ABS(M(R.I,R.K))
      THEN R.VM=ABS(M(R.I,R.K)) :
        R.IM=R.I
2545   NEXT
2550   IF R.VM<R.EPS
      THEN ER%=-14
      ELSE GOSUB 2585
2555   R.K=R.K+1
2560 WEND
2565 FOR R.I=0 TO R.N-1:
      FOR R.J=0 TO R.N-1:
        M(R.I,R.J)=R.M(R.I,R.J):
      NEXT:
NEXT
2570 ERASE R.M
2580 RETURN
2585 IF R.IM<>R.K
      THEN FOR R.J=0 TO R.N-1:
        R.AUX=M(R.IM,R.J):
        M(R.IM,R.J)=M(R.K,R.J):
        M(R.K,R.J)=R.AUX:
        R.AUX=R.M(R.IM,R.J):
        R.M(R.IM,R.J)=R.M(R.K,R.J):
        R.M(R.K,R.J)=R.AUX:
      NEXT
2590 R.DV=M(R.K,R.K)
2595 FOR R.J=0 TO R.N-1:
      M(R.K,R.J)=M(R.K,R.J)/R.DV:
      R.M(R.K,R.J)=R.M(R.K,R.J)
NEXT
2600 FOR R.I=0 TO R.N-1
2605   IF R.I<>R.K
      THEN R.M=M(R.I,R.K):
      FOR R.J=0 TO R.N-1:
        M(R.I,R.J)=M(R.I,R.J)-R.M*
        M(R.K,R.J):
        R.M(R.I,R.J)=R.M(R.I,R.J)+R.M*
        R.M(R.K,R.J):
      NEXT
2610 NEXT
2615 RETURN
MERGE "C7"
OK

```

```

MERGE "C9"
Ok
MERGE "S6"
Ok
RUN
Inversa unei matrice
N( 1 <= N <= 10 ) = 3
Linia 1 : 1, 2, 3,
Linia 2 : 2, 5, 2,
Linia 3 : 7, 3, 8,
Matricea inversa este :
-.596491 .122807 .192982
.0350877 .22807 -.070174
.508772 -.192982 -.017439
Continuati ? (Y/N) : N
Ok

```

Listing 5.34

5.5 Șiruri de caractere

După cum este cunoscut, unitatea de bază adresabilă în calculatoarele numerice este octetul (baitul). Cei 8 biți ai acestuia permit alcătuirea a 256 combinații binare (de la 00000000 la 11111111), ceea ce oferă posibilitatea reprezentării pe un octet a tot atâtea coduri. Ca atare există posibilitatea reprezentării nu numai a codurilor ASCII (anexa B) ci și a încă o serie de semne speciale.

Acest mod de utilizare a octetului dă posibilitatea interpretării informațiilor memorate în calculator nu numai ca numere ci și ca șiruri de caractere (de simboluri obținute prin combinarea diferită a biților aparținând unui octet). Astfel un caracter este asimilat cu un octet.

Atenție. Deoarece reprezentarea internă a informațiilor în calculator este unică (configurații binare), deosebirea și interpretarea corectă a acestora (valori numerice sau șiruri de caractere) revine utilizatorului. Pentru aceasta, așa cum s-a arătat în capitolul 3, programatorul are la dispoziție atît funcții cît și moduri de scriere a variabilelor diferite specifice celor două moduri de interpretare și tratare a informațiilor.

În acest subcapitol prezentăm patru aplicații deosebit de simple dar utile, mai ales începătorilor în programare, ca exemplificări ale unor aspecte specifice lucrului cu șiruri de caractere.

● Conversia unui număr natural ($32 \leq N \leq 26$) în caracter

Aplicația este prezentată în listingul 5.35.

```

10 PRINT "Conversii numar -> caracter"
20 C.MIN=32 : C.MAX=126 : C.MAT$="Nr." : C$="Y"
30 WHILE C$="Y" OR C$="y"
40 GOSUB 3180 : N=C.A
50 PRINT "Numar      caracter      hexazecimal      octal"
60 PRINT N, CHR$(N) , HEX$(N) , OCT$(N)

```

5. Aplicații

```

70 INPUT "Continuati ? (Y/N) : ",C$
80 WEND
90 END
MERGE "C7"
Ok
RUN
Conversie numar -> caracter
Nr.( 32 <= N <= 126 ) = 66
Numar caracter hexazecimal octal
66 B 42 102
Continuati ? (Y/N) : Y
Nr.( 32 <= N <= 126 ) = 125
Numar caracter hexazecimal octal
125 } 7D 175
Continuati ? (Y/N) : N
Ok

```

Listing 5.35

În afară de valoarea numerică introdusă și caracterul ASCII corespunzător acesteia, se afișează și transcrierea hexazecimală și octală corespunzătoare, în scopul facilitării înțelegerii celor două interpretări ale valorii binare memorate într-un octet.

De exemplu:

$$66_{(10)} = 102_{(8)} = 42_{(16)} = 1000010_{(2)}$$

dar configurația binară 01000010 (anexa B) corespunde codului ASCII al caracterului B. Deci interpretată ca valoare numerică secvența binară 01000010 reprezintă numărul zecimal 66, pe când caracterul reprezintă litera B.

● Crearea unui cap de tabel

Aplicația este prezentată în listingul 5.36.

```

10 PRINT "Creare cap tabel"
20 M$="M A T E R I I"
30 E$="= Numele si prenumele elevului |"
40 A$="| romana |matematica| fizica | chimie ="
50 CR$=CHR$(13)+CHR$(10)
60 PRINT STRING$(76,61);CR$;"=";SPC(30);"|" ;SPC(12);
M$;SPC(12);"=";CR$;E$;STRING$(43,45);"=";CR$;
"=";SPC(30);A$;CR$;"=";STRING$(74,45);"="
70 END
RUN
Creare cap tabel
=====
= Numele si prenumele elevului | M A T E R I I =
= | romana |matematica| fizica | chimie =
=====
Ok

```

Listing 5.36

● Conversia unui șir de cifre în valoare numerică

Aplicația, prezentată în listingul 5.37, verifică șirul introdus caracter cu caracter și, dacă este corect, efectuează conversia acestuia în număr zecimal. Conversia se efectuează cu funcția VAL (linia 110).

```

10 PRINT "Validare si conversie"
20 PRINT "sir de cifre -> valoare numerica"
30 C$="Y"
40 WHILE C$="Y" OR C$="y"
50 INPUT "NR :",N$
60 BUN=1
70 FOR I=1 TO LEN(N$)
80 A$=MID$(N$,I,1)
90 IF NOT (A$>"0" AND A$<="9") AND A$<> "."
    THEN BUN=0
    NEXT
110 IF BUN<>1
    THEN PRINT "Numar incorect".
    ELSE N=VAL(N$):
    PRINT "S-a tastat numarul :";N
120 INPUT "Continuati ? (Y/N) : ",C$
130 WEND
140 END
RUN

```

```

Validare si conversie
sir de cifre -> valoare numerica
NR :12.3
S-a tastat numarul : 12.3
Continuati ? (Y/N) : Y
NR :17P8
Numar incorect
Continuati ? (Y/N) : Y
NR :1708
S-a tastat numarul : 1708
Continuati ? (Y/N) : N
Ok

```

Listing 5.37

● Căutarea unui subsir într-un șir dat

Aplicația este prezentată în listingul 5.38.

```

10 PRINT "Cauta si localizeaza un subsir"
20 PRINT "intr-un sir dat"
30 C$="Y"
40 WHILE C$="Y" OR C$="y"
50 PRINT "Introduceti sirul : "
60 INPUT "",A$
70 INPUT "Subsir : ",B$
80 WHILE B$<>""
90 N=INSTR(1,A$,B$)
100 IF N=0
    THEN PRINT "Subsir inexistent"
    ELSE PRINT "Subsirul incepe la pozitia";N

```

```

110 INPUT "Subsir :",B$
120 WEND
130 INPUT "Continuati ? (Y/N) : ",C$
140 WEND
150 END
RUN
Cauta si localizeaza un subsir
intr-un sir dat
Introduceti sirul :
Acesta este sirul dat
Subsir :SIR
Subsir inexistent
Subsir :sir
Subsirul incepe la pozitia 13
Subsir :
Continuati ? (Y/N) : N
Ok
    
```

Listing 5.38

5.6 Fișiere

În capitolul 3, exemplul 3.19 a fost creat fișierul secvențial CLASA. Un program pentru listarea acestuia s-a prezentat în exemplul 3.20, fiind ulterior salvat în fișierul LIST. De asemenea a fost ilustrat modul de creare a unui fișier în acces direct (fișierul CLASAD - exemplul 3.21).

Subcapitolul de față prezintă două aplicații ce exemplifică modulele de actualizare ale fișierelor CLASA (aplicația 39) și CLASAD (aplicația 40).

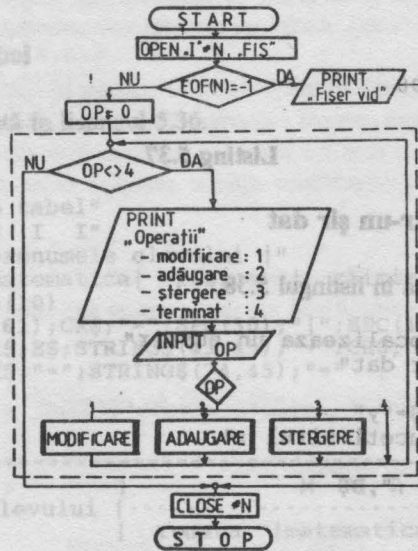


Fig. 5.26

Deoarece programele de actualizare a fişierelor sînt în general destul de complexe și uneori pun probleme, prezentăm în figura 5.26 o schemă logică principală de abordare a unor astfel de probleme.

După cum se poate observa, dacă deschiderea fişierului s-a efectuat cu succes, utilizatorul se află în fața unui meniu și, funcție de opțiunea OP, poate alege procedura de prelucrare dorită. La rîndul lor procedurile pot fi împărțite în module, astfel încît să se obțină o parcurgere arborescentă, clară și simplă a tuturor posibilităților (necesităților) de prelucrare. După parcurgerea procedurii alese se revine la meniul principal.

Remarcă. În lucrul cu fişiere în acces direct gestiunea articolelor (a numerelor înregistrărilor) scrise, șterse sau neutilizate rămîne în sarcina utilizatorului. Din această cauză actualizarea fişierelor în acces direct trebuie executată cu multă atenție deoarece există pericolul (în cazul utilizării neatențe a numerelor înregistrărilor) de distrugere a unor informații utile.

Observație. Ambele programe prezentate în aplicațiile 39 și 40 au fost create și testate în BASIC-80 (MBASIC,GBASIC) sub CP/M.

● Actualizarea fişierului secvențial CLASA

Aplicația este prezentată în listingul 5.39.

```

10 PRINT "Actualizarea fisierului secvențial CLASA"
20 C.MIN=1 : C.MAX=10
30 OPEN "I",#1,"CLASA"
40 IF EOF(1)=-1
    THENPRINT "Fisierul clasa este vid":
    ER%=-15
    ELSEOP=0:
    WHILE OP>4:
        GOSUB 80:
    WEND
50 IF ER%=0
    THEN PRINT "Actualizare Ok"
    ELSE PRINT "ER =",ER%
60 CLOSE #1
70 END
80 'Precizare operatie
90 PRINT "Operatii : "
100 PRINT " - modificare (1) "
110 PRINT " - adaugare (2) "
120 PRINT " - stergere (3) "
130 PRINT " - terminare (4) "
140 INPUT "Operatie (1/2/3/4) : ",OP
150 ON OP GOSUB 180,370,670
160 RETURN
170 'Modificare
180 INPUT "Nume : ",NUME$
190 WHILE NUME$<>" "
200 INPUT "Prenume : ",PREN$
210 ELEV$=NUME$+" "+PREN$
220 OPEN "O",#2,"MAN"

```

5. Aplicații

```

230     IND=0
240     WHILE EOF(1)<>-1
250     INPUT #1,E$,MR,MM,MF,MC
260     IF E$<>ELEV$
        THEN WRITE #2,E$,MR,MM,MF,MC
        ELSE GOSUB 870:
    IND=1
270     WEND
280     IF IND<>1
        THEN PRINT "Elev inexistent"
290     CLOSE #1
300     CLOSE #2
310     KILL "CLASA"
320     NAME "MAN" AS "CLASA"
330     OPEN "I",#1,"CLASA"
340     INPUT "Nume      :",NUME$
350     WEND
360     RETURN
370     'Adaugare
380     OPEN "O",#2,"MAN"
390     WHILE EOF(1)<>-1
400     INPUT #1,E$,MR,MM,MF,MC
410     WRITE #2,E$,MR,MM,MF,MC
420     WEND
430     CLOSE #1
440     INPUT "Nume      :",NUME$
450     WHILE NUME$<>""
460     INPUT "Prenume : ",PREN$
470     ELEV$=NUME$+" "+PREN$
480     C.MAT$="Romana      "
490     GOSUB 3180 : MR=C.A
500     C.MAT$="Matematica"
510     GOSUB 3180 : MM=C.A
520     C.MAT$="Fizica      "
530     GOSUB 3180 : MF=C.A
540     C.MAT$="Chimie     "
550     GOSUB 3180 : MC=C.A
560     PRINT ELEV$;" : ";MR;MM;MF;MC
570     INPUT "Corect ? (Y/N) : ",CD$
580     IF CD$="Y" OR CD$="y"
        THEN WRITE #2,ELEV$,MR,MM,MF,MC
    INPUT "Nume      :",NUME$
590     WEND
600     CLOSE #2
610     KILL "CLASA"
620     NAME "MAN" AS "CLASA"
630     OPEN "I",#1,"CLASA"
640     PRINT "Adaugare Ok"
650     RETURN
660     'Stergere
670     INPUT "Nume      :",NUME$
680     WHILE NUME$<>""
690     INPUT "Prenume : ",PREN$
700     ELEV$=NUME$+" "+PREN$
710     OPEN "O",#2,"MAN"
720     INDA=0
730     WHILE EOF(1)<>-1
740     INPUT #1,E$,MR,MM,MF,MC
750     IF E$<>ELEV$
        THEN WRITE #2,E$,MR,MM,MF,MC

```



```

ELSE INDA=1:
GOSUB 1040:
INPUT "Stergeti ? (Y/N) : ",CD$:
IF CD$="Y" OR CD$="y"
THEN PRINT "Articol sters"
ELSE WRITE #2,E$,MR,MM,MF,MC:
PRINT "Articol salvat"
770 WEND
780 IF INDA<>1
THEN PRINT "Elev inexistent"
790 CLOSE #1
800 CLOSE #2
810 KILL "CLASA"
820 NAME "MAN" AS "CLASA"
830 OPEN "I",#1,"CLASA"
840 INPUT "Nume : ",NUME$
850 WEND
860 RETURN
870 Modificare medii
880 INDA=0
890 WHILE INDA<>1
900 GOSUB 1040
910 C.MAT$="Romana "
920 GOSUB 3180 : MR=C.A
930 C.MAT$="Matematica"
940 GOSUB 3180 : MM=C.A
950 C.MAT$="Fizica "
960 GOSUB 3180 : MF=C.A
970 C.MAT$="Chimie "
980 GOSUB 3180 : MC=C.A
990 PRINT ELEV$;" : ";MR;MM;MF;MC
1000 INPUT "Corect ? (Y/N) : ",CD$
1010 IF CD$="Y" OR CD$="y"
THEN WRITE #2,ELEV$,MR,MM,MF,MC:
PRINT "Modificare Ok":
INDA=1
1020 WEND
1030 RETURN
1040 'Afisare elev
1050 PRINT "Elev : ",E$
1060 PRINT " romana matematica fizica chimie"
1070 PRINT USING " ## " ;MR;MM;MF;MC
1080 RETURN
SAVE "ACT",A
Ok
LOAD "LIST",R
Listarea fisierului secvential CLASA
Numele si prenumele elevului romana matematica fizica chimie
Alexe Ion 7 8 9 9
Coman Lili 6 7 6 7
Dobre Nicu 7 8 8 9
Ok
LOAD "ACT",R
Actualizarea fisierului secvential CLASA
Operatii :
- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)
Operatie (1/2/3/4) : 2

```

5. Aplicații

Nume :Florea
 Prenume :Gheorghe
 Romana (1 <= N <= 10) = 9
 Matematica (1 <= N <= 10) = 10
 Fizica (1 <= N <= 10) = 9
 Chimie (1 <= N <= 10) = 9
 Florea Gheorghe : 9 10 9 9
 Corect ? (Y/N) : Y

Nume :Nicolau
 Prenume :Carmen
 Romana (1 <= N <= 10) = 8
 Matematica (1 <= N <= 10) = 9
 Fizica (1 <= N <= 10) = 8
 Chimie (1 <= N <= 10) = 8
 Florea Gheorghe : 8 8 9 8
 Corect ? (Y/N) : Y

Nume :
 Adaugare Ok

Operatii :

- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)

Operatie (1/2/3/4) : 1

Nume :Coman

Prenume :Lili

Elev : Coman Lili

	romana	matematica	fizica	chimie
	6	7	6	7

Romana (1 <= N <= 10) = 9
 Matematica (1 <= N <= 10) = 10
 Fizica (1 <= N <= 10) = 10
 Chimie (1 <= N <= 10) = 9

Coman Lili : 9 10 10 9

Corect ? (Y/N) : Y

Modificare Ok

Nume :

Operatii :

- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)

Operatie (1/2/3/4) : 3

Nume :Dobre

Prenume :Nicu

Elev : Dobre Nicu

	romana	matematica	fizica	chimie
	7	8	8	9

Stergeti ? (Y/N) : Y

Articol sters

Nume :

Operatii :

- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)

Operatie (1/2/3/4) : 4

Actualizare Ok

Ok

LOAD "LIST",R

Lista	romana	matematica	fizica	chimie
Alexe Ion	7	8	9	9
Coman Lili	9	10	10	9
Florea Gheorghe	9	10	9	9
Nicolau Carmen	8	9	8	8

Listing 5.39

● Actualizarea fişierului cu acces direct CLASAD

Aplicaţia este prezentată în listingul 5.40.

```

10 PRINT "Actualizarea fişierului"
20 PRINT "cu acces direct CLASAD"
30 C.MIN=1 : C.MAX=10
40 OPEN "R",#1,"CLASAD",40
50 FIELD #1,28 AS EL$,3 AS MR$,3 AS MM$,3 AS MF$,3 AS MC$
60 OP=0
70 WHILE OP<>4
80 PRINT "Operatii :"
90 PRINT " - modificare (1)"
100 PRINT " - inserare (2)"
110 PRINT " - stergere (3)"
120 PRINT " - terminare (4)"
130 INPUT "Operatie (1/2/3/4) : ",OP
140 ON OP GOSUB 180,280,360
150 WEND
160 PRINT "Actualizare Ok"
170 END
180 'Modificare
190 INPUT "Numar inregistrare : ",NR%
200 WHILE NR%>0
210 GET #1,NR%
220 GOSUB 550
230 GOSUB 590
240 INPUT "Numar inregistrare : ",NR%
250 WEND
260 PRINT "Modificare Ok"
270 RETURN
280 'Inserare
290 INPUT "Numar inregistrare : ",NR%
300 WHILE NR%>0
310 GOSUB 630
320 INPUT "Numar inregistrare : ",NR%
330 WEND
340 PRINT "Inserare Ok"
350 RETURN
360 'Stergere
370 INPUT "Numar inregistrare : ",NR%
380 WHILE NR%>0
390 GET #1,NR%
400 GOSUB 550
410 INPUT "Stergeti articolul ? (Y/N) : ",CD$
420 IF CD$="Y" OR CD$="y"
    THEN GOSUB 470:
    PRINT "Articol sters"

```

5. Aplicații

```

ELSE PRINT "Articol salvat"
430 INPUT "Numar inregistrare : ",NR%
440 WEND
450 PRINT "Stergere Ok"
460 RETURN
470 '
480 LSET EL$=SPACE$(28)
490 LSET MR$=SPACE$(3)
500 LSET MF$=SPACE$(3)
510 LSET MM$=SPACE$(3)
520 LSET MC$=SPACE$(3)
530 PUT #1,NR%
540 RETURN
550 'Afișare
560 PRINT "Elev : ";EL$
570 PRINT "romana:";MR$;"          matematica:";MM$;
    " fizica:";MF$;"          chimie:";MC$
580 RETURN
590 '
600 INPUT "Modificati articolul ? (Y/N) : ",CD$
610 IF CD$="Y" OR CD$="y"
    THEN GOSUB 630
620 RETURN
630 'Introducere articol
640 INPUT "Nume      : ",NUME$
650 INPUT "Prenume   : ",PREN$
660 ELEV$=NUME$+" "+PREN$
670 C.MAT$="Romana  "
680 GOSUB 3180 : MR=C.A
690 C.MAT$="Matematica"
700 GOSUB 3180 : MM=C.A
710 C.MAT$="Fizica  "
720 GOSUB 3180 : MF=C.A
730 C.MAT$="Chimie  "
740 GOSUB 3180 : MC=C.A
750 PRINT ELEV$;" : ";MR;MM;MF;MC
760 INPUT "Articol corect ? (Y/N) : ",C$
770 IF C$="Y" OR C$="y"
    THEN LSET EL$=ELEV$ :
        RSET MR$=STR$(MR) :
        RSET MM$=STR$(MM) :
        RSET MF$=STR$(MF) :
        RSET MC$=STR$(MC) :
        PUT #1,NR%
780 RETURN
MERGE "C7"
OK
RUN
Actualizarea fisierului
cu acces direct CLASAD
Operatii :
- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)
Operatie (1/2/3/4) : 1
Numar inregistrare : 3
Elev : Luca Ion
romana: 9 matematica: 10 fizica: 9 chimie: 9
Modificati articolul ? (Y/N) : Y

```

```

Nume      :Luca
Prenume   :Ion
Romana    ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 10
Fizica    ( 1 <= N <= 10 ) = 9
Chimie    ( 1 <= N <= 10 ) = 8
Florea Gheorghe : 9 10 9 8
Articol corect ? (Y/N) : Y

```

```

Numar inregistrare :
Modificare Ok

```

```

Operatii :
- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)

```

```
Operatie (1/2/3/4) : 2
```

```
Numar inregistrare : 4
```

```

Nume      :Rusu
Prenume   :Emil
Romana    ( 1 <= N <= 10 ) = 8
Matematica( 1 <= N <= 10 ) = 7
Fizica    ( 1 <= N <= 10 ) = 7
Chimie    ( 1 <= N <= 10 ) = 7
Florea Gheorghe : 8 7 7 7
Articol corect ? (Y/N) : Y

```

```

Numar inregistrare :
Inserare Ok

```

```

Operatii :
- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)

```

```
Operatie (1/2/3/4) : 3
```

```
Numar inregistrare : 5
```

```

Elev : Toader Virgil
romana: 9 matematica: 10 fizica: 8 chimie: 8

```

```
Stergeti articolul ? (Y/N) : Y
```

```

Articol sters
Numar inregistrare :

```

```

Stergere Ok
Operatii :
- modificare (1)
- adaugare (2)
- stergere (3)
- terminare (4)

```

```
Operatie (1/2/3/4) : 4
```

```
Actualizare Ok
```

```
OK
```

Listing 5.40

5. Aplicații

```

Name : Luca
Program : Ion
Romans ( 1 <- N <- 10 ) = 3
Mathematical ( 1 <- N <- 10 ) = 10
Physics ( 1 <- N <- 10 ) = 9
Chemistry ( 1 <- N <- 10 ) = 8
Physics George ( 2 10 2 )
Article correct ? (Y/N) : Y
Number integers :
Modifier OK
Operator :
Operator (1)
- add (2)
- subtract (3)
- terminate (4)
Operator (1/2/3/4) : 2
Number integers : 4
Name : Kana
Program : Emi
Romans ( 1 <- N <- 10 ) = 3
Mathematical ( 1 <- N <- 10 ) = 7
Physics ( 1 <- N <- 10 ) = 7
Chemistry ( 1 <- N <- 10 ) = 7
Physics George ( 2 7 7 )
Article correct ? (Y/N) : Y
Number integers :
Integer OK
Operator :
Operator (1)
- add (2)
- subtract (3)
- terminate (4)
Operator (1/2/3/4) : 3
Number integers : 5
Name : Todor Virgil
Name : Todor Virgil
Romans : 2
Mathematical : 10
Article correct ? (Y/N) : Y
Article state
Number integers :
Integer OK
Operator :
Operator (1)
- add (2)
- subtract (3)
- terminate (4)
PUT #1, NR3
RETURN
Name : C7
Program :
Operator OK
Operator (1/2/3/4) : 4
Number integers : 4
Name : Luca Ion
Romans : 9
Mathematical : 10
Physics : 8
Chemistry : 8
Article correct ? (Y/N) : Y

```

Listing 2.40

ANEXA A

PRINCIPALELE CARACTERISTICI ALE MODULELOR BASIC UTILIZATE ÎN EXEMPLELE PREZENTATE

Fișier rutină	Numere linii	Funcția realizată	Variabile			ER% ¹⁾	Ape- luri	Lstg. defi- nire
			Intrare	Ieșire	Globale			
R1	1000-1010	Împărțirea cu rest ($N2 < > 0$)			N2, N1	-1		5.1
R2	1020-1060	CMMDC al două numere naturale diferite de zero	R.A	R.C		-2		4.9
R3	1065-1096	CMMDC al două numere întregi	R.A R.B	R.C		-3 -4		5.3
R4	1100-1115	CMMDC al două numere întregi	R.A R.B	R.C		-3 -4	R3	5.4
R5	1120-1135	CMMDC al R.N numere întregi ($2 \leq R.N \leq 20$)	R.N	R.C	A(20)	-5	R3	5.5
R7	1180-1280	CMMDC al R.N numere întregi ($2 \leq R.N \leq 20$)	R.N	R.C	A(20)	-5	R3	5.6
R8	1300-1375	Determinarea numerelor prime $\leq R.N$ ($2 \leq R.N \leq 1000$)	R.N		PRIM(300)	-6		5.7
R9	1380-1420	Convertește R.N din baza 10 în baza R.B ($0 \leq R.N \leq 32767$; $2 \leq R.B \leq 9$)	R.N R.B		CIFRE(16) ORD	-3		5.9
R10	1440-1470	Convertește în baza 10 un număr de 16 cifre binare	R.A		CIFRE(16)	-8 -9		5.11
R11	1480-1565	Descompunerea unui număr întreg în factori primi	R.N	R.NR	F(100) P(100)	-3		5.12
R12	1590-1655	Determină minimumul ($R.I=0$) sau maximumul ($R.I < > 0$) și locul său în V0; ($1 \leq R.N \leq 20$)	R.N R.I	R.M R.L	V(20)	-2		5.13

R13	1660-1675	Suma elementelor vectorului VO ($1 \leq R.N \leq 20$)	R.N	R.S	V(20)	-2	5.14
R14	1680-1730	Caută în VO elementul R.NR ($1 \leq R.N \leq 20$)	R.NR	R.L	V(20)	-2	5.15
R15	1740-1810	Detremină numărul elementelor negative nule și pozitive din VO; ($1 \leq R.N \leq 20$)	R.N	R.NG R.Z R.P	V(20)	-2	5.16
R16	1820-1830	Schimbă pozițiile a două elemente ale lui VO între ele ($1 \leq R.N \leq 20$)	R.N R.A R.B		V(20)	-2	5.17
R17	1840-1850	Inversează ordinea elementelor lui VO; ($1 \leq R.N \leq 20$)	R.N		V(20)	-2	5.18
R18	1860-1930	Ordonarea crescătoare / descrescătoare a elementelor lui VO ($2 \leq R.N \leq 20$)	R.N R.I		V(20)	-5	5.20
R19	1940-1950	Mută R.K elemente de la începutul la sfârșitul lui VO ($2 \leq R.N \leq 20$)	R.N R.K		V(20)	-2 -5	5.19
R20	1960-2045	Rotirea stînga ($R.I > 0$) sau dreapta ($R.I < 0$) a elementelor lui VO; ($2 \leq R.N \leq 20$)	R.N R.I		V(20)	-2 -5	5.21
R21	2050-2125	Deplasarea stînga ($R.I > 0$) sau dreapta ($R.I < 0$) a elementelor lui VO; ($2 \leq R.N \leq 20$)	R.N R.I		V(20)	-2 -5	5.22
R22	2130-2170	Verifică dacă VO este o mulțime ($1 \leq R.N \leq 10$)	R.N		V(10)	-10 -11	5.23
R23	2175-2220	Reuniunea mulțimilor A0 și B0 ($1 \leq R.N, R.M \leq 10$)	R.N R.M	R.K	A(10) B(10) C(20)	-10 -11	5.24
R24	2230-2265	Intersecția mulțimilor A0 și B0 ($1 \leq R.N, R.M \leq 10$)	R.N R.M	R.K	A(10) B(10) C(10)	-10 -11	5.25

R25	2280-2325	Diferența mulțimilor $A \setminus B$ ($1 \leq R.N, R.M \leq 10$)	R.N R.M	R.K	A(10) B(10) C(20)	-10 -11	5.26
R26	2330-2450	Calculul valorii unui determinant ($1 \leq R.N \leq 10$; R.EPS ≥ 0.0001)	R.EPS R.N	R.D	M(10,10)	-12 -13	5.33
R27	2460-2615	Inversa unei matrice ($1 \leq R.N \leq 10$; R.EPS ≥ 0.0001)	R.N R.EPS		M(10,10)	-12 -13 -14	5.34
C1	3000-3030	Citește și validează $N1, N2$ *			N1, N2		4.9
C2	3040-3055	Citește un număr întreg		C.A			5.2
C3	3060-3095	Citește un vector de maxim 20 de numere întregi			N, A(20)		5.5
C4	3100-3130	Citește un număr binar de maxim 16 cifre ($1 \leq C.A \leq 16$)	C.A		CIFRE (16)	-6	5.8
C5	3135-3150	Citește și validează un număr natural diferit de zero		C.A			5.13
C6	3155-3175	Citește un șir de C.A numere în V0; ($1 \leq C.A \leq 20$)	C.A		V(20)	-2	5.13
C7	3180-3200	Citește și validează un număr întreg cuprins între C.MIN și C.MAX	C.MIN C.MAX C.MAT\$	C.A			3.19
C8	3210-3260	Citește și validează o mulțime de R.N elemente		R.N	V(10)	C(7) R18 R22	5.23
C9	3270-3280	Citește o matrice de C.A x C.B elemente ($1 \leq C.A, C.B \leq 10$)	C.A C.B		M(10,10)	-12	5.28
S1	5000-5010	Afișează CMMDC			CMMDC		4.6
S2	5020-5030	Afișează CMMMC			CMMMC		4.9

S3	5040-5090	Determină și afișează divizorii unui număr întreg	R.N				5.8
S4	5100-5145	Afișează un vector de S.N valori numerice (1≤S.N≤20)	S.N S.V\$		V(20)		5.17
S5	5150-5195	Afișează componentele unei mulțimi VO; (1≤S.N≤20)	S.N S.V\$		V(20)		5.24
S6	5200-5210	Afișează elementele matricei M() (1≤S.A,S.B≤10)	S.A S.B		M(10,10)	-12	5.28

d) Coduri de eroare returnate de module (ER%):

- 0 execuție normală (fără erori)
- 1 împărțire la zero
- 2 N1 sau N2 în afara limitelor
- 3 N1 sau N2 sînt numere întregi
- 4 în determinarea lui CMMDC, toate numerele sînt nule
- 5 parametru mai mic decît 2 sau mai mare decît 20
- 6 parametru mai mic decît 2 sau mai mare decît 1000
- 7 baza mai mică decît 2 sau mai mare decît 9
- 8 ORD<0 sau ORD>15
- 9 într-un număr binar s-au detectat cifre diferite mai mari decît 1
- 10 șir neordonat sau element dublu definit într-o mulțime
- 11 parametru mai mic decît 1 sau mai mare decît 10
- 12 parametru în afara limitelor în R26 sau R27
- 13 |R.EPS|<10⁻⁴
- 14 matrice singulară
- 15 fișier vid

ANEXA B

Codul ASCII

Zecimal	Binar	Octal	Hexa	Caracter
000	0000000	000	00	<NUL>
001	0000001	001	01	<SOH> (Start of heading)
002	0000010	002	02	<STX> (Start of text)
003	0000011	003	03	<ETX> (End of text)
004	0000100	004	04	<END> (End of transmission)
005	0000101	005	05	<ENQ> (Enquiry)
006	0000110	006	06	<ACK> (Acknowledge)
007	0000111	007	07	<BELL>
008	0001000	010	08	<BS> (Backspace)
009	0001001	011	09	<HT> (Horizontal tabulation)
010	0001010	012	0A	<LF> (Line feed)
011	0001011	013	0B	<VT> (Vertical tabulation)
012	0001100	014	0C	<FF> (Form feed)
013	0001101	015	0D	<CR> (Carriage return)
014	0001110	016	0E	<SO> (Shift out)
015	0001111	017	0F	<SI> (Shift in)
016	0010000	020	10	<DLE> (Data link espace)
017	0010001	021	11	<DC1> (Device control 1)
018	0010010	022	12	<DC2> (Device control 2)
019	0010011	023	13	<DC3> (Device control 3)
020	0010100	024	14	<DC4> (Device control 4)
021	0010101	025	15	<NAK> (Negative aknowledge)
022	0010110	026	16	<SYN> (Synchronous idle)

023	0010111	027	17	<ETB> (End of transmission block)
024	0011000	028	18	<CAN> (Cancel)
025	0011001	031	19	 (End of medium)
026	0011010	032	1A	<SUB> (Substitute)
027	0011011	033	1B	<ESCAPE>
028	0011100	034	1C	<FS> (File separator)
029	0011101	035	1D	<GS> (Group separator)
030	0011110	036	1E	<RS> (Record separator)
031	0011111	037	1F	<US> (Unit separator)
032	0100000	040	20	<SPACE>
033	0100001	041	21	!
034	0100010	042	22	"
035	0100011	043	23	#
036	0100100	044	24	\$
037	0100101	045	25	%
038	0100110	046	26	&
039	0100111	047	27	'
040	0101000	050	28	(
041	0101001	51	29)
042	0101010	52	2A	*
043	0101011	53	2B	+
044	0101100	054	2C	,
045	0101101	055	2D	-
046	0101110	056	2E	.
047	0101111	057	2F	/
048	0110000	060	30	0
049	0110001	061	31	1
050	0110010	062	32	2

051	0110011	063	33	3
052	0110100	064	34	4
053	0110101	065	35	5
054	0110110	066	36	6
055	0110111	067	37	7
056	0111000	070	38	8
057	0111001	071	39	9
058	0111010	072	3A	:
059	0111011	073	3B	;
060	0111100	074	3C	<
061	0111101	075	3D	=
062	0111110	076	3E	>
063	0111111	077	3F	?
064	1000000	100	40	@
065	1000001	101	41	A
066	1000010	102	42	B
067	1000011	103	43	C
068	1000100	104	44	D
069	1000101	105	45	E
070	1000110	106	46	F
071	1000111	107	47	G
072	1001000	110	48	H
073	1001001	111	49	I
074	1001010	112	4A	J
075	1001011	113	4B	K
076	1001100	114	4C	L
077	1001101	115	4D	M
078	1001110	116	4E	N

079	1001111	117	4F	O
080	1010000	120	50	P
081	1010001	121	51	Q
082	1010010	122	52	R
083	1010011	123	53	S
084	1010100	124	54	T
085	1010101	125	55	U
086	1010110	126	56	V
087	1010111	127	57	W
088	1011000	130	58	X
089	1011001	131	59	Y
090	1011010	132	5A	Z
091	1011011	133	5B	[
092	1011100	134	5C	\
093	1011101	135	5D]
094	1011110	136	5E	^
095	1011111	137	5F	_
096	1100000	140	60	·
097	1100001	141	61	a
098	1100010	142	62	b
099	1100011	143	63	c
100	1100100	144	64	d
101	1100101	145	65	e
102	1100110	146	66	f
103	1100111	147	67	g
104	1101000	150	68	h
105	1101001	151	69	i
106	1101010	152	6A	j

107	1101011	153	6B	k
108	1101100	154	6C	l
109	1101101	155	6D	m
110	1101110	156	6E	n
111	1101111	157	6F	o
112	1110000	160	70	p
113	1110001	161	71	q
114	1110010	162	72	r
115	1110011	163	73	s
116	1110100	164	74	t
117	1110101	165	75	u
118	1110110	166	76	v
119	1110111	167	77	w
120	1111000	170	78	x
121	1111001	171	79	y
122	1111010	172	7A	z
123	1111011	173	7B	{
124	1111100	174	7C	
125	1111101	175	7D	}
126	1111110	176	7E	~
127	1111111	177	7F	 (Delete)

1990

[14] Vidova I., Beltac V., Florescu V., Floridica L., Jitaru M. - 'Ingineria programării', Editura Academiei R.S.R., 1985

[15] Zaharia M. - 'FORTRAN-77. Depanarea programelor', Editura Tehnică, București 1991

[16] *** - 'Dicționar de informatică', Editura Științifică și Enciclopedică, București 1991

[17] *** - 'Calculatoare electronice din generația a cincea', Editura Academiei R.S.R., București 1985

[18] *** - 'BETA BASIC' V3.0, Manual de utilizare, Editura 1985

Să învățăm să programăm

101	0101011	151	69	l			
102	0101011	152	69	l			
103	0001011	151	69	l			
104	0001011	151	69	l			
105	0001011	151	69	l			
106	1110011	147	79	g			
107	0110011	148	79	g			
108	1010011	148	79	g			
109	0010011	144	69	p			
110	1100011	141	69	o			
111	1100011	141	69	o			
112	1111111	137	79	h			
113	1111111	137	79	h			
114	1111110	136	79	h			
115	1111101	132	79	h			
116	1111100	134	79	h			
117	1111011	133	79	h			
118	1111010	133	79	h			
119	1111001	131	79	h			
120	1111000	130	79	h			
121	1110111	123	79	h			
122	1110110	123	79	h			
123	1110101	124	79	h			
124	1110100	124	79	h			
125	1101111	127	79	h			
126	1101110	126	79	h			
127	1101101	127	79	h			
128	1101100	124	79	h			
129	1101011	123	79	h			
130	1101010	123	79	h			
131	1101001	121	79	h			
132	1101000	120	79	h			
133	1011111	117	49	Q			
134	1011110	117	49	Q			
135	1011101	117	49	Q			
136	1011100	117	49	Q			
137	1011011	117	49	Q			
138	1011010	117	49	Q			
139	1011001	117	49	Q			
140	1011000	117	49	Q			
141	1010111	117	49	Q			
142	1010110	117	49	Q			
143	1010101	117	49	Q			
144	1010100	117	49	Q			
145	1010011	117	49	Q			
146	1010010	117	49	Q			
147	1010001	117	49	Q			
148	1010000	117	49	Q			
149	1001111	117	49	Q			
150	1001110	117	49	Q			
151	1001101	117	49	Q			
152	1001100	117	49	Q			
153	1001011	117	49	Q			
154	1001010	117	49	Q			
155	1001001	117	49	Q			
156	1001000	117	49	Q			
157	1000111	117	49	Q			
158	1000110	117	49	Q			
159	1000101	117	49	Q			
160	1000100	117	49	Q			
161	1000011	117	49	Q			
162	1000010	117	49	Q			
163	1000001	117	49	Q			
164	1000000	117	49	Q			
165	0111111	117	49	Q			
166	0111110	117	49	Q			
167	0111101	117	49	Q			
168	0111100	117	49	Q			
169	0111011	117	49	Q			
170	0111010	117	49	Q			
171	0111001	117	49	Q			
172	0111000	117	49	Q			
173	0110111	117	49	Q			
174	0110110	117	49	Q			
175	0110101	117	49	Q			
176	0110100	117	49	Q			
177	0110011	117	49	Q			
178	0110010	117	49	Q			
179	0110001	117	49	Q			
180	0110000	117	49	Q			
181	0101111	117	49	Q			
182	0101110	117	49	Q			
183	0101101	117	49	Q			
184	0101100	117	49	Q			
185	0101011	117	49	Q			
186	0101010	117	49	Q			
187	0101001	117	49	Q			
188	0101000	117	49	Q			
189	0100111	117	49	Q			
190	0100110	117	49	Q			
191	0100101	117	49	Q			
192	0100100	117	49	Q			
193	0100011	117	49	Q			
194	0100010	117	49	Q			
195	0100001	117	49	Q			
196	0100000	117	49	Q			
197	0011111	117	49	Q			
198	0011110	117	49	Q			
199	0011101	117	49	Q			
200	0011100	117	49	Q			
201	0011011	117	49	Q			
202	0011010	117	49	Q			
203	0011001	117	49	Q			
204	0011000	117	49	Q			
205	0010111	117	49	Q			
206	0010110	117	49	Q			
207	0010101	117	49	Q			
208	0010100	117	49	Q			
209	0010011	117	49	Q			
210	0010010	117	49	Q			
211	0010001	117	49	Q			
212	0010000	117	49	Q			
213	0001111	117	49	Q			
214	0001110	117	49	Q			
215	0001101	117	49	Q			
216	0001100	117	49	Q			
217	0001011	117	49	Q			
218	0001010	117	49	Q			
219	0001001	117	49	Q			
220	0001000	117	49	Q			
221	0000111	117	49	Q			
222	0000110	117	49	Q			
223	0000101	117	49	Q			
224	0000100	117	49	Q			
225	0000011	117	49	Q			
226	0000010	117	49	Q			
227	0000001	117	49	Q			
228	0000000	117	49	Q			
229	0111111	117	49	Q			
230	0111110	117	49	Q			
231	0111101	117	49	Q			
232	0111100	117	49	Q			
233	0111011	117	49	Q			
234	0111010	117	49	Q			
235	0111001	117	49	Q			
236	0111000	117	49	Q			
237	0110111	117	49	Q			
238	0110110	117	49	Q			
239	0110101	117	49	Q			
240	0110100	117	49	Q			
241	0110011	117	49	Q			
242	0110010	117	49	Q			
243	0110001	117	49	Q			
244	0110000	117	49	Q			
245	0101111	117	49	Q			
246	0101110	117	49	Q			
247	0101101	117	49	Q			
248	0101100	117	49	Q			
249	0101011	117	49	Q			
250	0101010	117	49	Q			
251	0101001	117	49	Q			
252	0101000	117	49	Q			
253	0100111	117	49	Q			
254	0100110	117	49	Q			
255	0100101	117	49	Q			
256	0100100	117	49	Q			
257	0100011	117	49	Q			
258	0100010	117	49	Q			
259	0100001	117	49	Q			
260	0100000	117	49	Q			
261	0011111	117	49	Q			
262	0011110	117	49	Q			
263	0011101	117	49	Q			
264	0011100	117	49	Q			
265	0011011	117	49	Q			
266	0011010	117	49	Q			
267	0011001	117	49	Q			
268	0011000	117	49	Q			
269	0010111	117	49	Q			
270	0010110	117	49	Q			
271	0010101	117	49	Q			
272	0010100	117	49	Q			
273	0010011	117	49	Q			
274	0010010	117	49	Q			
275	0010001	117	49	Q			
276	0010000	117	49	Q			
277	0001111	117	49	Q			
278	0001110	117	49	Q			
279	0001101	117	49	Q			
280	0001100	117	49	Q			
281	0001011	117	49	Q			
282	0001010	117	49	Q			
283	0001001	117	49	Q			
284	0001000	117	49	Q			
285	0000111	117	49	Q			
286	0000110	117	49	Q			
287	0000101	117	49	Q			
288	0000100	117	49	Q			
289	0000011	117	49	Q			
290	0000010	117	49	Q			
291	0000001	117	49	Q			
292	0000000	117	49	Q			
293	0111111	117	49	Q			
294	0111110	117	49	Q			
295	0111101	117	49	Q			
296	0111100	117	49	Q			
297	0111011	117	49	Q			
298	0111010	117	49	Q			
299	0111001	117	49	Q			
300	0111000	117	49	Q			
301	0110111	117	49	Q			
302	0110110	117	49	Q			
303	0110101	117	49	Q			
304	0110100	117	49	Q			
305	0110011	117	49	Q			
306	0110010	117	49	Q			
307	0110001	117	49	Q			
308	0110000	117	49	Q			
309	0101111	117	49	Q			
310	0101110	117	49	Q			
311	0101101	117	49	Q			
312	0101100	117	49	Q			
313</							

Bibliografie

- [1] Dodescu Gh., Ionescu D., Nisipeanu L., Pilat F. - 'Limbajul BASIC și aplicații', Editura Didactică și Pedagogică, București 1978
- [2] Dumitrașcu L., Ioachim Al. - 'Tehnici de construire a programelor cu structuri alternative', Editura Academiei R.S.R., 1981
- [3] Dumitrașcu L. - 'BASIC pentru începători cu calculatorul personal', A.M.C. Nr. 48, Editura Tehnică, București 1986
- [4] Dumitrașcu L. - 'Învățăm microelectronica interactivă', Editura Tehnică, București 1989
- [5] Dobrescu P. - 'Computere și trandafiri sau paradoxurile progresului', Editura Politică, București 1988
- [6] Iorgulescu A. - 'Exerciții de programare structurată în FORTRAN', Atelierul poligrafic ASE, București 1980
- [7] Moldovan G. - 'Scheme logice și programe FORTRAN', Editura Didactică și Pedagogică, București 1978
- [8] Moraru F., Atodiroaei M. - 'Programarea microcalculatoarelor în sistemul de operare CP/M', Editura Științifică și Enciclopedică, București 1989
- [9] Niculescu St., Dumitrescu M. - 'Algoritmi și metode de reprezentare', Editura Tehnică, București 1978
- [10] Niculescu St. - 'FORTRAN inițiere în programarea structurată' Editura Tehnică, București 1979
- [11] Petrescu A. ș.a. - 'Totul despre ... calculatorul personal aMIC', Editura Tehnică, București 1984
- [12] Petrescu A. ș.a. - 'Microcalculatoarele Felix M18, M118B, M118', Editura Tehnică, București 1984
- [13] Petrescu A. ș.a. - 'ABC de calculatoare personale și ... nu doar atât ...', Editura Tehnică, București 1990
- [14] Văduva I., Baltac V., Florescu V., Floricica I., Jitaru M. - 'Ingineria programării', Editura Academiei R.S.R., 1985
- [15] Zaharia M. - 'FORTRAN-77. Depanarea programelor', Editura Tehnică, București 1991
- [16] * * * - 'Dicționar de informatică', Editura Științifică și Enciclopedică, București 1981
- [17] * * * - 'Calculatoare electronice din generația a cincea', Editura Academiei R.S.R., București 1985
- [18] * * * - 'BETA BASIC' V3.0, Manual de utilizare, Betasoft 1985

Copyright © 1992, Editura Tehnică
Toate drepturile asupra acestei ediții
sînt rezervate editurii

Adresa: Editura Tehnică
Piața Presei Libere, 1
33 București, România
cod 79738

Redactor: ing. Silvia Cîndea
Coperta: grafician Simona Dumitrescu
Tehnoredactare: ing. Silvia Cîndea

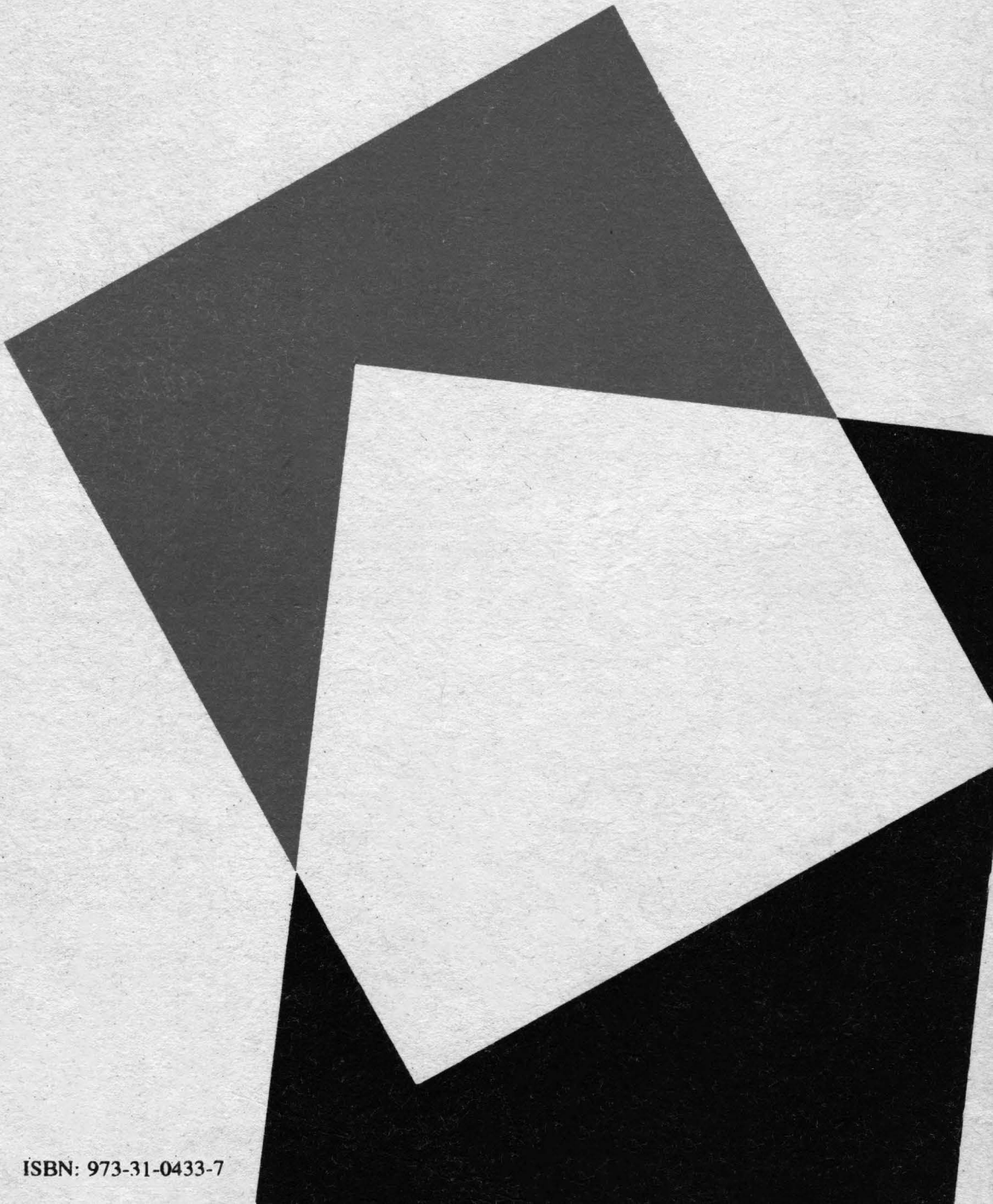
Coli de tipar 12; Bun de tipar: 15.1.92

C.Z.: 681.3.06

ISBN: 973-31-0433-7

Tiparul executat sub comanda nr. 1987
la IMPRIMERIA "ARDEALUL" CLUJ

Lei 210,-



ISBN: 973-31-0433-7